SMART TRAFFIC LIGHT CONTROL SYSTEM

BY

ANKIT SAXENA

AJAY KUMAR

BHARAT BHARDWAJ

MUSKAN MUSKAN

PRIYA R

1.Project Overview:

Description:

- The “**Smart Traffic Light Control System**” is an intelligent traffic solution designed to manage vehicle flow at intersections with two crossing roads. Unlike traditional fixed-timer systems, it adapts signal timings in real-time based on actual traffic conditions.

- **Ultrasonic sensors** are installed on each road to detect the number of waiting vehicles. This data is sent to the STM32F405RGT6 microcontroller, which processes it and dynamically adjusts the green light duration—giving priority to the road with more traffic. This helps reduce waiting time, fuel consumption, and congestion.

- To secure system access, a 4x4 matrix keypad is used, requiring a passkey (default: 1234) to activate the controller. A 16x2 LCD display shows system status and prompts, while a 4-digit 7-segment display shows countdown timers for signal changes, helping drivers prepare to move or stop.

- This smart, adaptive system improves traffic efficiency, reduces delays, and enhances safety—making it ideal for modern, smart city intersections.

2.Hardware and Software Requirements:

Component Type	Name / Version	Purpose
Microcontroller	STM32F405RGT6	➢ Serves as the main processing unit. It controls all input/output operations, processes sensor data, drives display modules, and manages signal timing.
Sensor	HC-SR04 Ultrasonic Sensor (x2)	➢ Used for real-time vehicle detection at the traffic lanes. The sensors measure the distance to vehicles and help optimize signal timing based on traffic density.
Display	7-Segment 4-Digit LED	➢ Displays a countdown timer to inform drivers and pedestrians of the time remaining for each traffic signal.
Input	4x4 Keypad	➢ Allows users (e.g., operators or maintenance staff) to input a numeric passkey to access or configure the system.
Signal Actuators	Traffic Light Module (Red, Yellow, Green)	➢ Controls the visual traffic signals per lane, indicating stop, get ready, or go for vehicles.
Power Supply	5V Adapter	➢ Provides regulated 5V DC supply to power all low-voltage components in the system such as microcontroller and sensors.
Power Module	DC Barrel Jack Power Supply Module	➢ Facilitates safe and stable DC power input from external adapters to the breadboard or circuit.

Wiring	Jumper Wires	➢ Used to make electrical connections between modules and components during prototyping.
Prototyping Board	Breadboard	➢ Enables non-permanent construction of the circuit, ideal for development and testing phases.
Software IDE	STM32CubeIDE	➢ Provides an integrated development environment for writing, compiling, and uploading firmware code to the STM32 microcontroller.
Diagnostic Tool	Multimeter	➢ Used to measure voltage, current, and resistance for diagnosing and debugging hardware components during development.

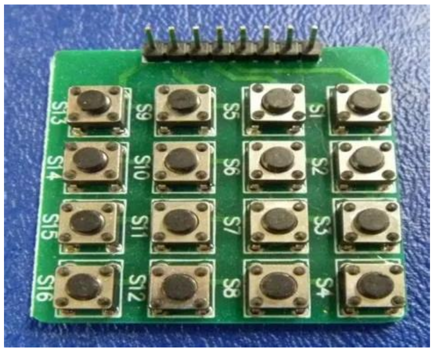
COMPONENTS:

1.STM32F405FRGT6 Microcontroller

2.LED Traffic Lights

3.Ultrasonic Sensor

4. 4x4 Matrix Keypad



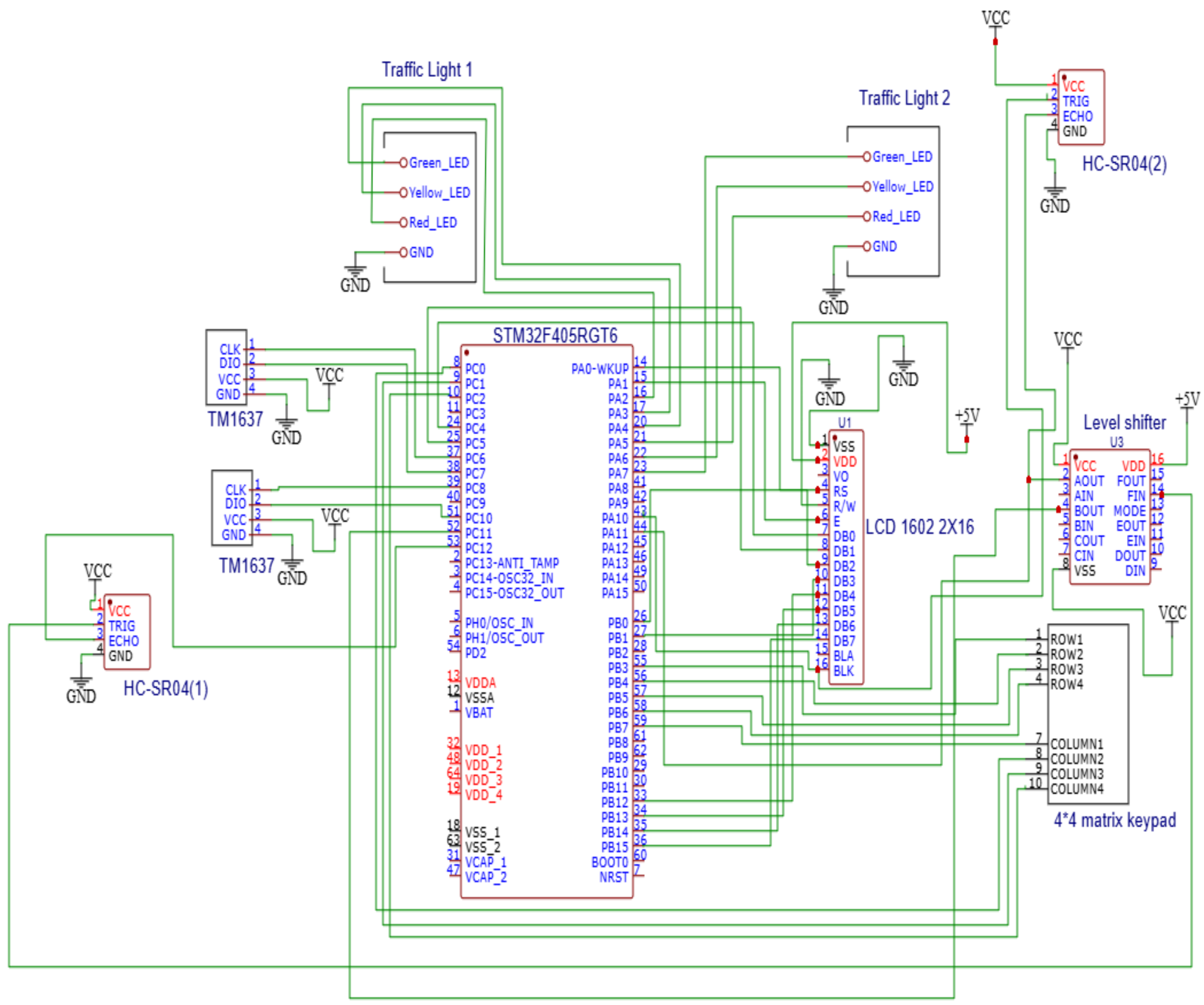
5. Multimeter:

Description:

- <u>**STM32F405RGT6**:</u>
  Acts as the **main controller**. It runs the traffic light logic, handles sensor input, keypad control, and manages timing.
- <u>**LED Traffic Light Module**:</u>
  Simulates **red, yellow, and green traffic lights** for two lanes Controlled by STM32 to direct traffic flow.
- <u>**HC-SR04 Ultrasonic Sensor**:</u>
  Detects the **presence or absence of vehicles** in a lane using ultrasonic waves. Helps in smart decision-making.
- <u>**Multimeter**:</u>
  Used for **measuring voltage, current, and continuity** while assembling and troubleshooting the circuit.
- <u>**4-Digit 7-Segment LED Display:**</u>
  Displays a **countdown timer** for vehicles or pedestrians, showing how much time is left for red/green signal.
- <u>**5V Adapter**:</u>
  Provides regulated **5V DC power** to the entire system. Powers the STM32 and connected modules.
- <u>**4×4 Keypad**:</u>
  Allows **manual input** to control or override the system (e.g., pedestrian request or manual signal change).
- <u>**Jumper Wires**:</u>
  Used for **connecting components** on the breadboard to the STM32 board and other modules.
- <u>**Breadboard**:</u>
  A **solderless platform** to build and test the circuit layout for the traffic control system.
- <u>**DC Barrel Jack Power Supply Module**:</u>
  Allows safe connection of the **5V adapter to the breadboard** and distributes power to components.

- **<u>Buzzer</u>:**
  Used for **audible alerts**, such as a pedestrian crossing warning or timer beeps.

3.Circuit Diagram:



- This circuit is a smart traffic control system designed using the STM32F405RGT6 microcontroller, which serves as the central processing unit for managing traffic lights based on real-time vehicle detection and user inputs.
- The system integrates multiple modules, including ultrasonic sensors (HC-SR04), 7-segment displays (TM1637), two sets of traffic lights, a 16x2 LCD, a 4x4 matrix keypad, and a level shifter for voltage compatibility.
- Two ultrasonic sensors are placed on each lane to detect vehicle presence. Sensor 1 uses PC11 for the TRIG signal and PC12 for the ECHO, while Sensor 2 uses PA10 for TRIG and PA11 for ECHO. Since the ECHO pins output 5V signals, a level shifter is used to convert this to 3.3V logic suitable for the STM32.
- The TM1637-based 7-segment displays are used to show countdown timers for the traffic signals, with one display connected to PC6 and PC7 and the other to PC8 and PC10. The traffic lights for two directions are controlled using PA2 to PA7, where each direction has red, yellow, and green LEDs controlled via separate pins.
- A 16x2 LCD is connected to the STM32 to display system status, with control pins RS and EN on PA0 and PA1, and data pins from D0 to D7 mapped to various GPIOs (PC4, PC5, PB0, PB1, PB12–PB15). The LCD is interfaced via a level shifter to accommodate its 5V operation.
- A 4x4 matrix keypad provides manual user input for controlling or configuring the system, with rows connected to PB3–PB6 and columns to PB7, PC0, PC1, and PC2. The level shifter ensures safe interfacing between 3.3V logic (STM32) and 5V devices.
- The STM32 is powered through its VDD and VDDA pins with a +5V supply, and appropriate capacitors are connected to the VCAP pins for internal voltage regulation. Overall, this setup creates an intelligent traffic management system capable of adapting signal timing based on vehicle detection, displaying real-time countdowns, and allowing user interaction for manual control or configuration.

1.Ultrasonic Sensor (HC-SR04) Connections

Sensor	Signal	STM32F405 Pin	Notes
#1	TRIG	**PC11**	Output to YF08E
#1	ECHO	**PC12**	Input via YF08E (3.3V)
#2	TRIG	**PA10**	Output to YF08E
#2	ECHO	**PA11**	Input via YF08E

2.7-Segment Display (TM1637) Connections

Display	CLK Pin	DIO Pin	STM32F405 Pins
#1	**CLK**	**DIO**	**PC6, PC7**
#2	**CLK**	**DIO**	**PC8, PC10**

3.Traffic Light Module Connections

LED Color	Module	STM32F405 Pin
RED	**Way 1**	**PA2**
YELLOW	**Way 1**	**PA3**
GREEN	**Way 1**	**PA4**
RED	**Way 2**	**PA5**
YELLOW	**Way 2**	**PA6**
GREEN	**Way 2**	**PA7**

4.LCD (16x2) Display Connections

LCD Pin	Function	STM32F405 Pin
RS	Control	**PA0**
EN	Enable	**PA1**
D0	Data Bit 0	**PC4**

D1	Data Bit 1	**PC5**
D2	Data Bit 2	**PB0**
D3	Data Bit 3	**PB1**
D4	Data Bit 4	**PB12**
D5	Data Bit 5	**PB13**
D6	Data Bit 6	**PB14**
D7	Data Bit 7	**PB15**
RW	Write Only	**GND**
VCC	Power	**+5V**
GND	Ground	**GND**

5.4x4 Matrix Keypad Connections

Function	**Keypad Pin**	**STM32F405 Pin**
Row 1	R1	**PB3**
Row 2	R2	**PB4**
Row 3	R3	**PB5**
Row 4	R4	**PB6**
Col 1	C1	**PB7**
Col 2	C2	**PC0**
Col 3	C3	**PC1**
Col 4	C4	**PC2**

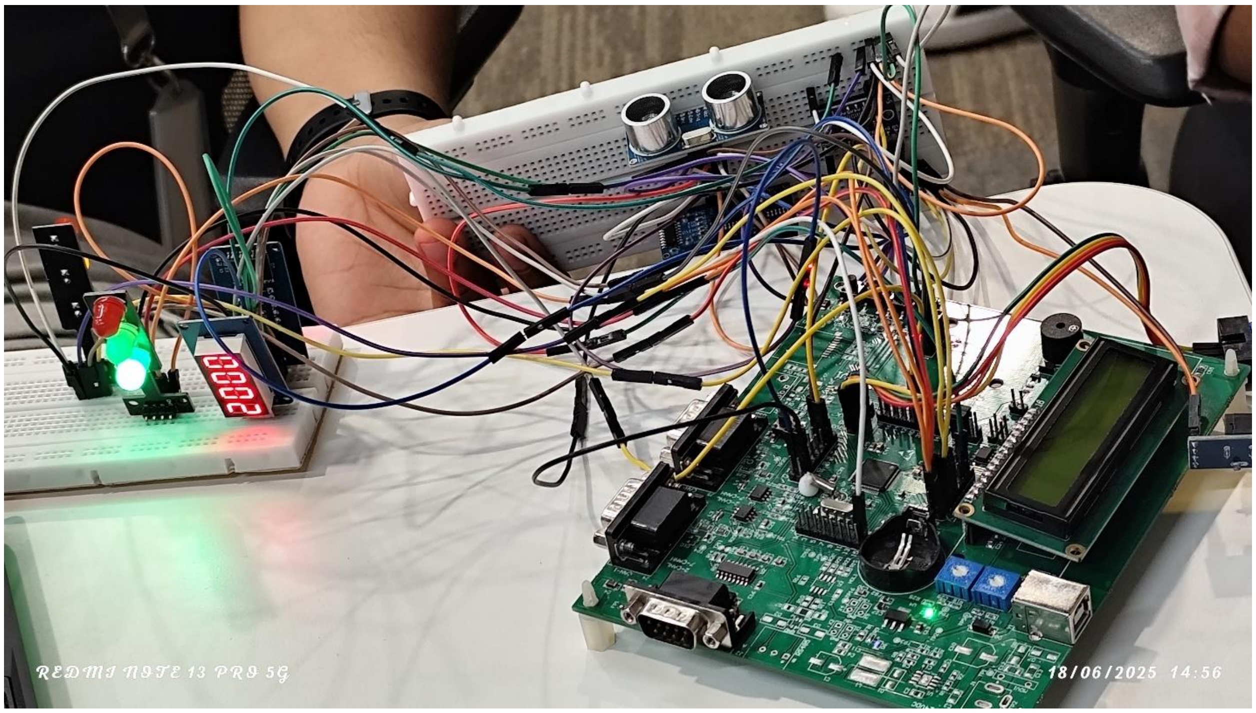
6. Level Shifter Connections

Pin Name	**Voltage Source**	**Purpose**
VCC	3.3V (STM32 logic)	Input side for STM32 logic level
VOD	5V (LCD side)	Output side to support LCD voltage
GND	Common ground	Shared ground for logic conversion

7.Power Supply Pins

STM32 Pin	**Connected To**	**Purpose**
VDD / VDDA	+5V Supply	Power input
VSS / VSSA	GND	Ground
VCAP	Capacitor	Internal voltage ref.

4.Hardware setup and Working:



Step-by-Step Connection:

- ❖ The two-way traffic light control system is designed to manage vehicle flow in two directions using an STM32F405RGT6 microcontroller. The system uses two HC-SR04 ultrasonic sensors to detect the presence of vehicles at both lanes. When a vehicle is detected in one lane, the microcontroller prioritizes that direction by turning on the green light for a fixed period, typically 10 seconds, while the opposite lane remains red. After the green period ends, the yellow light is activated for a short transitional period (usually 2–3 seconds) before switching the green light to the other lane if a vehicle is present there.

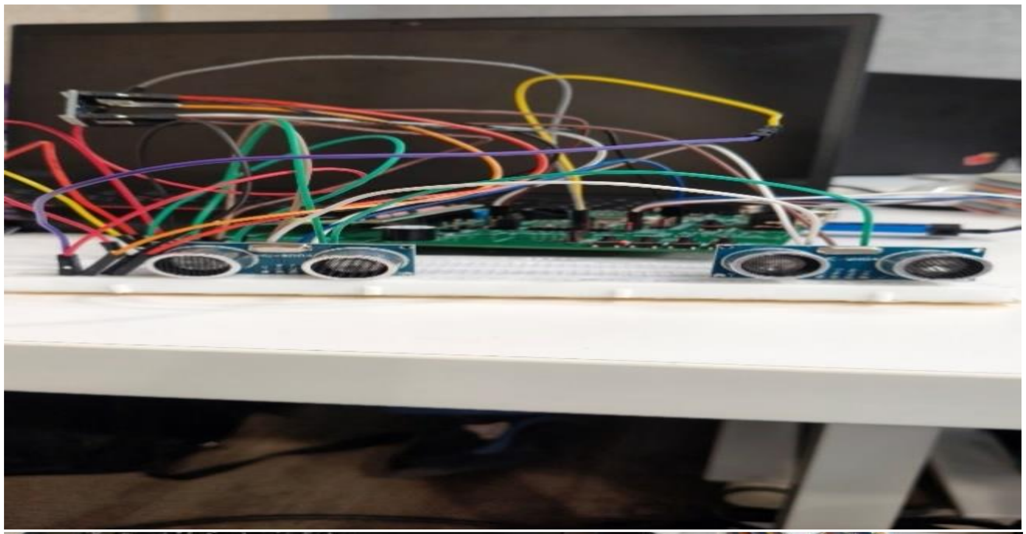
- ❖ A 7-segment 4-digit LED display shows a countdown timer indicating how long the green light will remain active. This display is updated every second through multiplexed GPIO control. Additionally, a 4x4 keypad is included to allow manual override or mode selection, giving operators the ability to switch between automatic and manual traffic control or adjust the duration of traffic light phases.

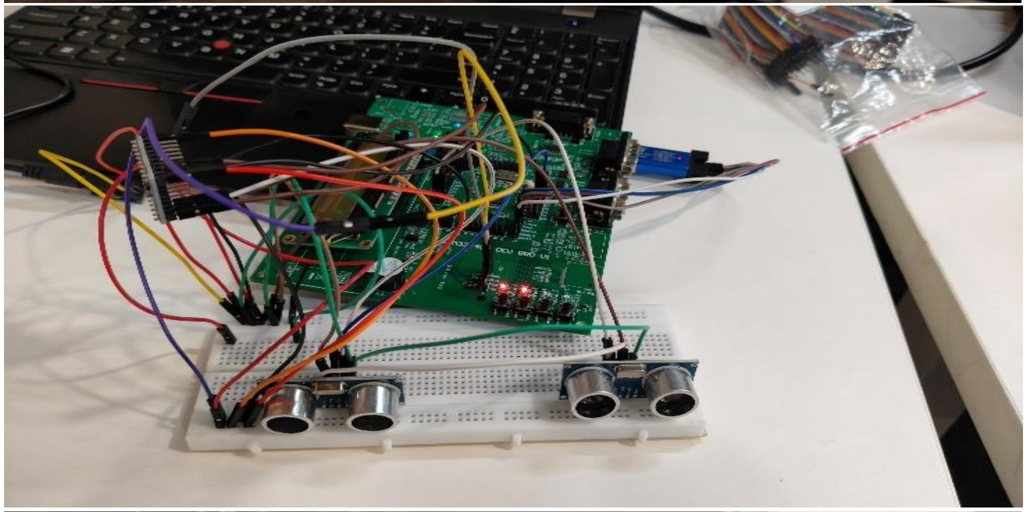
- ❖ The STM32 operates at 3.3V logic levels, while the HC-SR04 sensors use 5V signals. To ensure compatibility, a logic level shifter is used to safely connect the 5V Echo output from each sensor to the STM32's 3.3V GPIO pins. The traffic light modules consist of red, yellow, and green LEDs for each lane and are connected to the STM32 through current-limiting resistors.

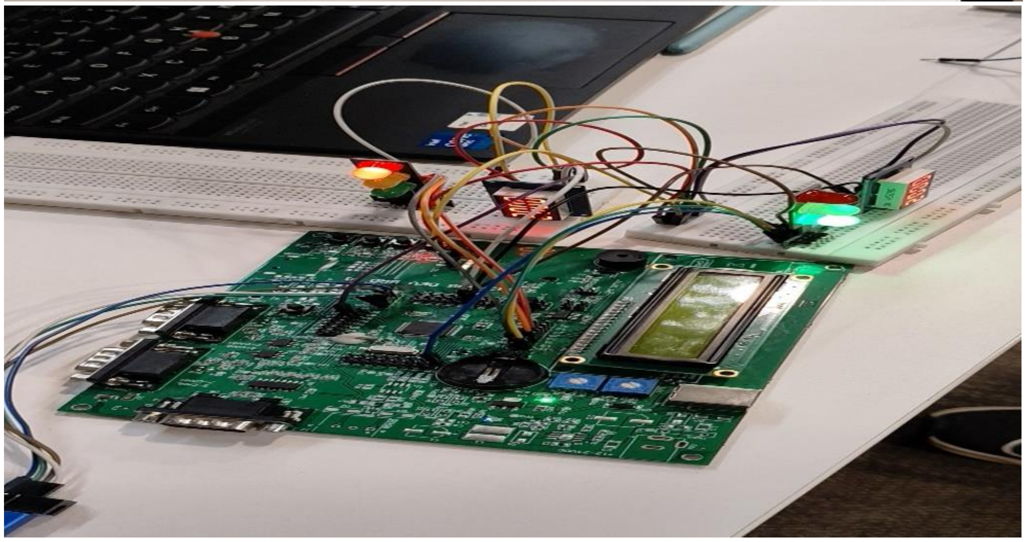
- ❖ Power is supplied through a 5V adapter connected to a DC barrel jack power supply module, which distributes 5V to high-voltage components and uses a voltage regulator to step down to 3.3V for the STM32. All components share a common ground to ensure proper circuit operation. The entire system is developed and programmed using STM32CubeIDE, which allows for configuring peripherals, timers, and GPIOs to handle sensor input, display output, and traffic light control efficiently.

- ❖ In summary, this system dynamically controls traffic flow based on vehicle presence using ultrasonic sensors and automates signal switching while providing visual feedback through a display and optional manual control via a keypad.

UNIT TESTING:

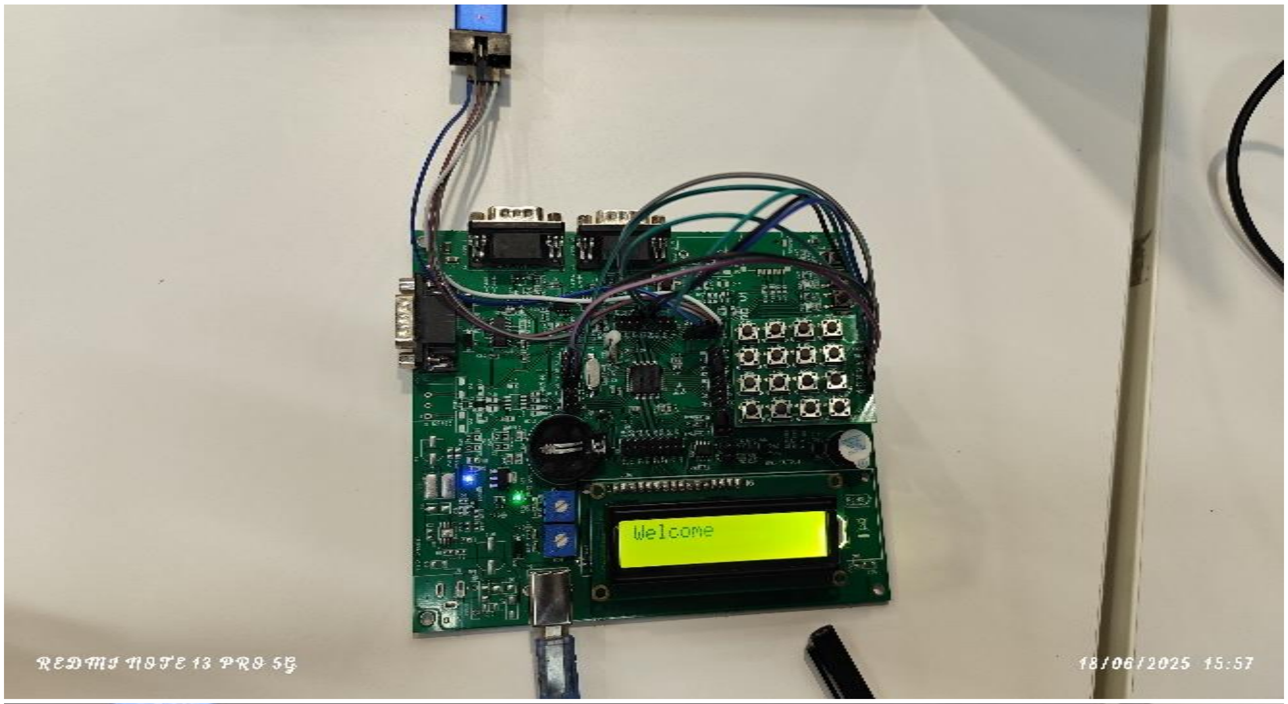


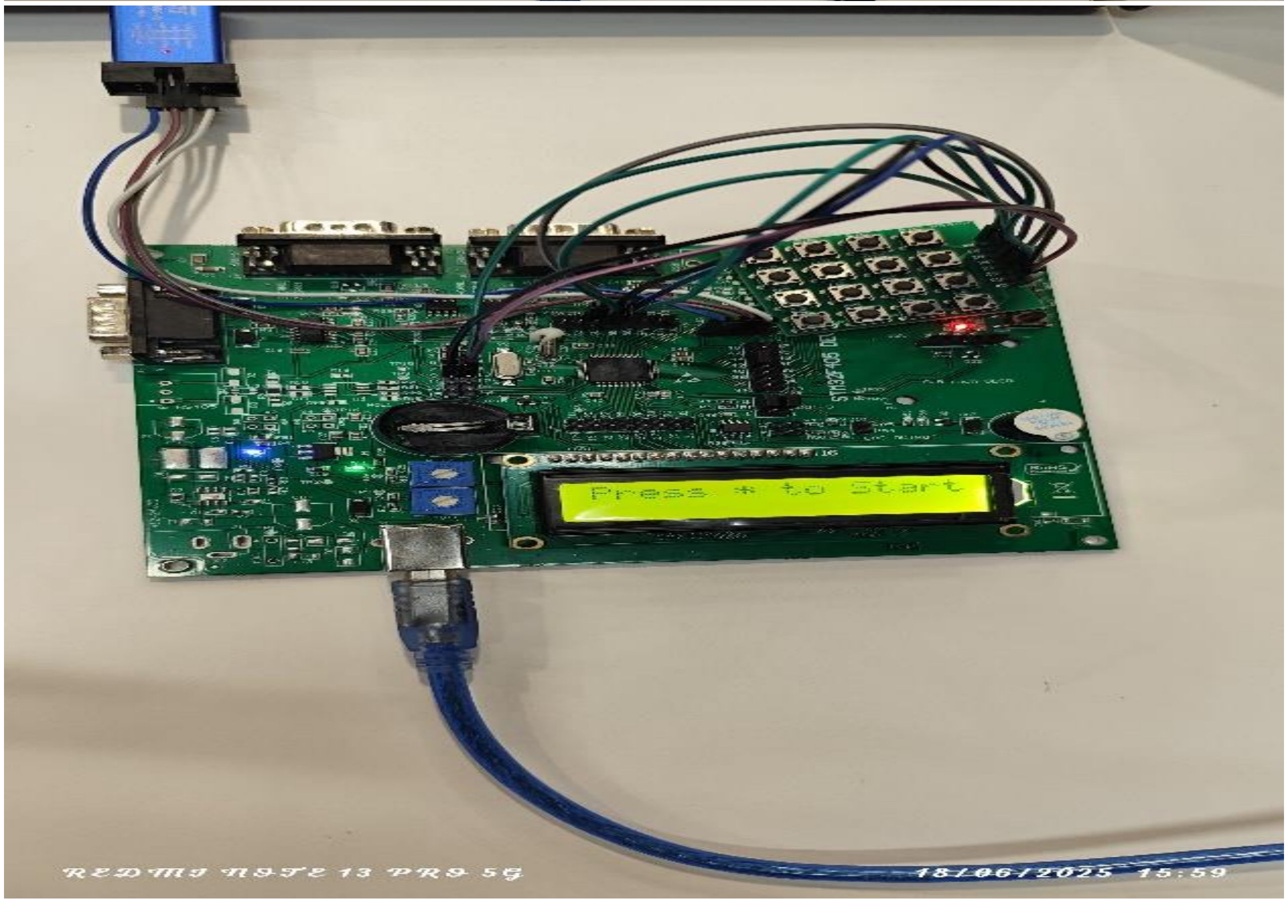


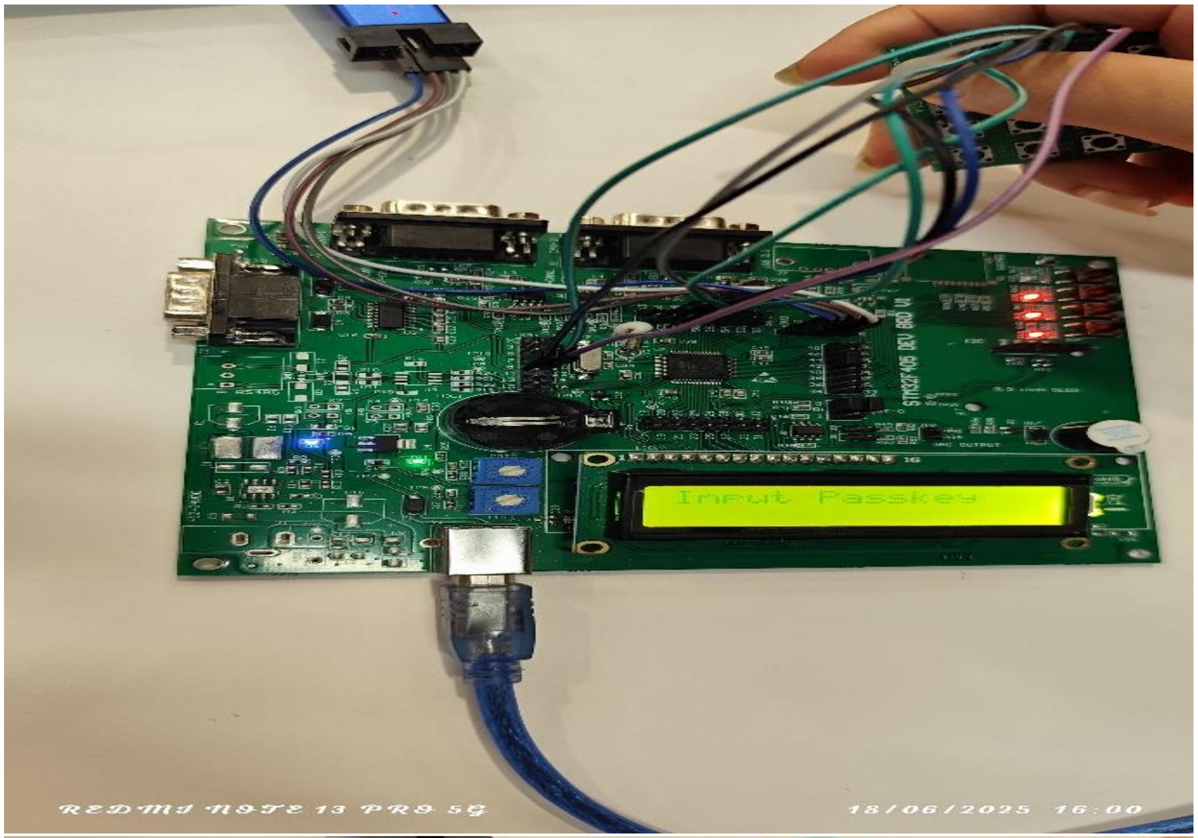


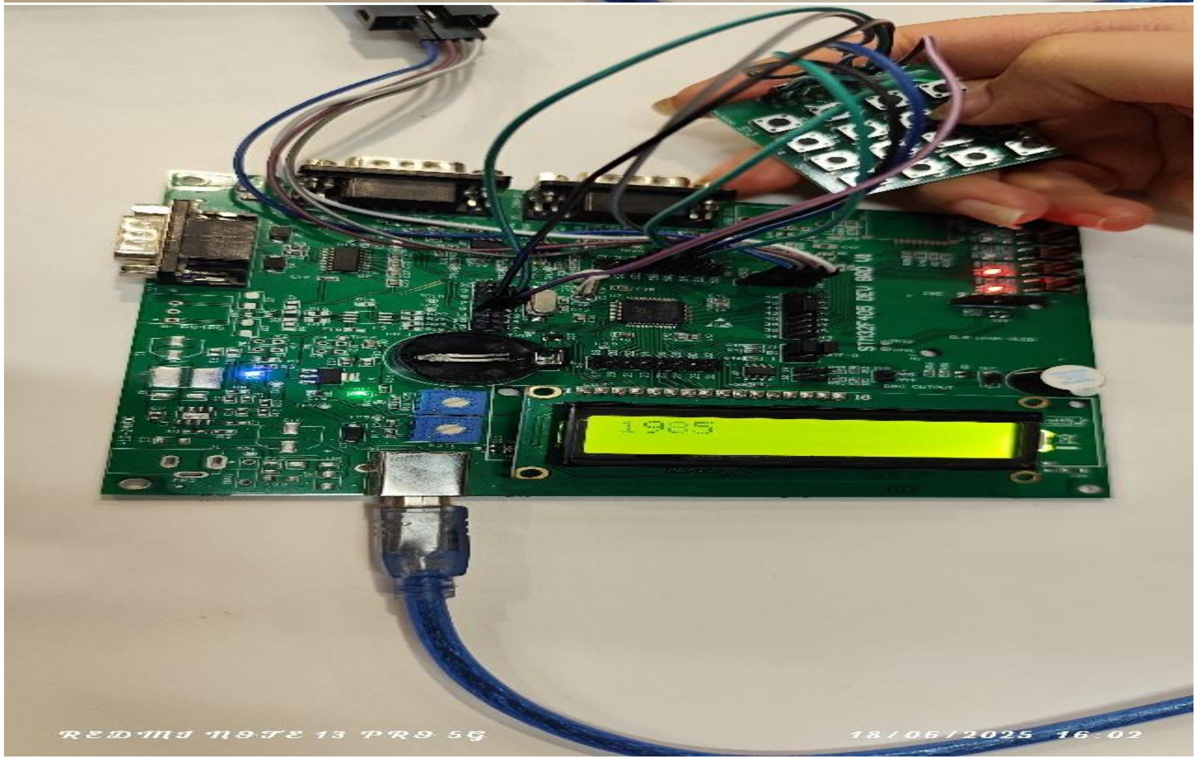
Unit	**Test Method**	**Pass Criteria**	**Tools Used**
LCD	Power ON and observe startup message	Displays: "Welcome"	Multimeter, Visual Check
Keypad	Press * → prompt, # after passkey	Displays response & toggles mode/lights	Oscilloscope, Visual Check
Ultrasonic Sensor A	Place object within 30 cm on Lane A	Triggers green light & countdown	Ruler, Logic Analyzer
Ultrasonic Sensor B	Place object within 30 cm on Lane B	Triggers green light & countdown	Ruler, Logic Analyzer
7-Segment Display	Observe countdown during green phase	Displays consistent decrement (e.g., 10 → 0)	Visual Check
Traffic Light A	Simulate A-cycle (Green → Yellow → Red)	Proper sequence with correct duration	Visual + Timer
Traffic Light B	Simulate B-cycle (Green → Yellow → Red)	Proper sequence with correct duration	Visual + Timer
Level Shifter	Input 5V signal → check 3.3V output	Output within 3.2–3.4V	Multimeter, Oscilloscope
Power Supply	Measure voltage at 3.3V & 5V lines	3.3V ±0.1V; 5V ±0.2V	Multimeter
STM32 GPIOs	Toggle output pins (blink test)	Pins switch HIGH/LOW as programmed	Multimeter, Oscilloscope
System Reset	Press reset button during operation	System restarts and displays initial state	Visual, Button Press
Buzzer/Alert	Trigger emergency condition manually	Audible beep or indicator activation	Visual, Audio Check
Manual Override Mode	Enter manual via keypad, set light manually	Light switches as per user input	Keypad, Visual Check
Sensor Calibration	Measure known distances and check response	Distance → consistent trigger within ±5%	Ruler, Logic Analyzer
Countdown Sync	Start sensor and watch display + light change	Countdown and light sync correctly	Stopwatch, Visual

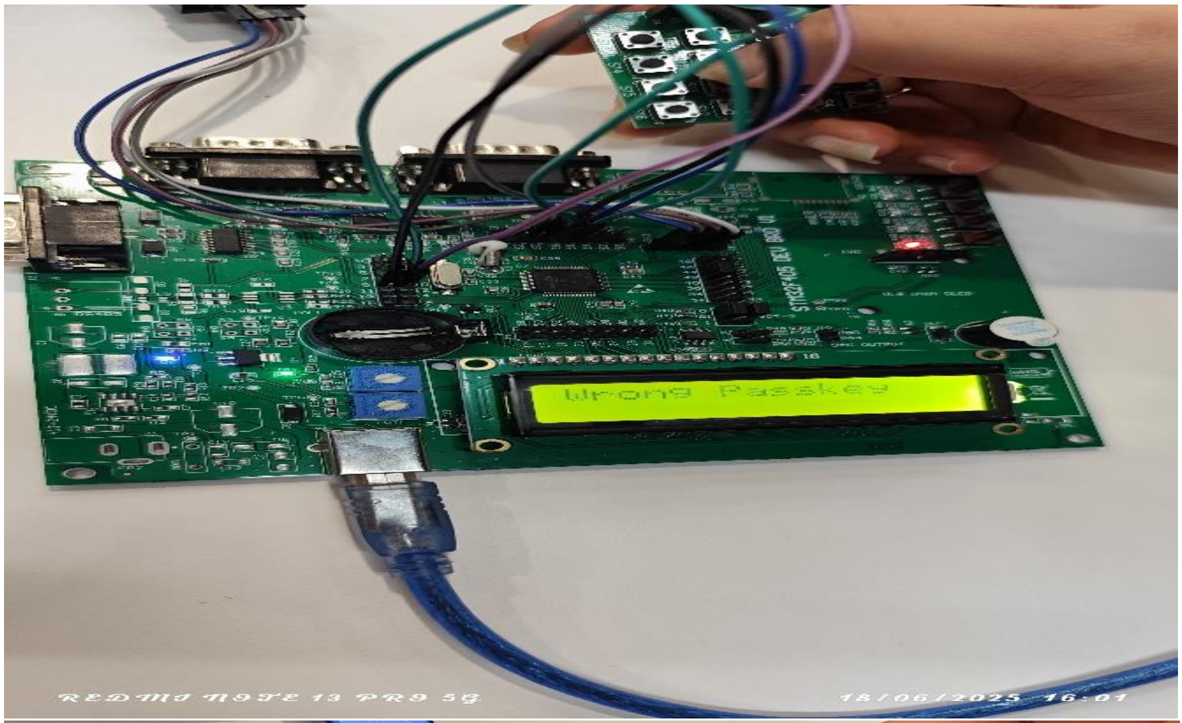
5.Test Cases:

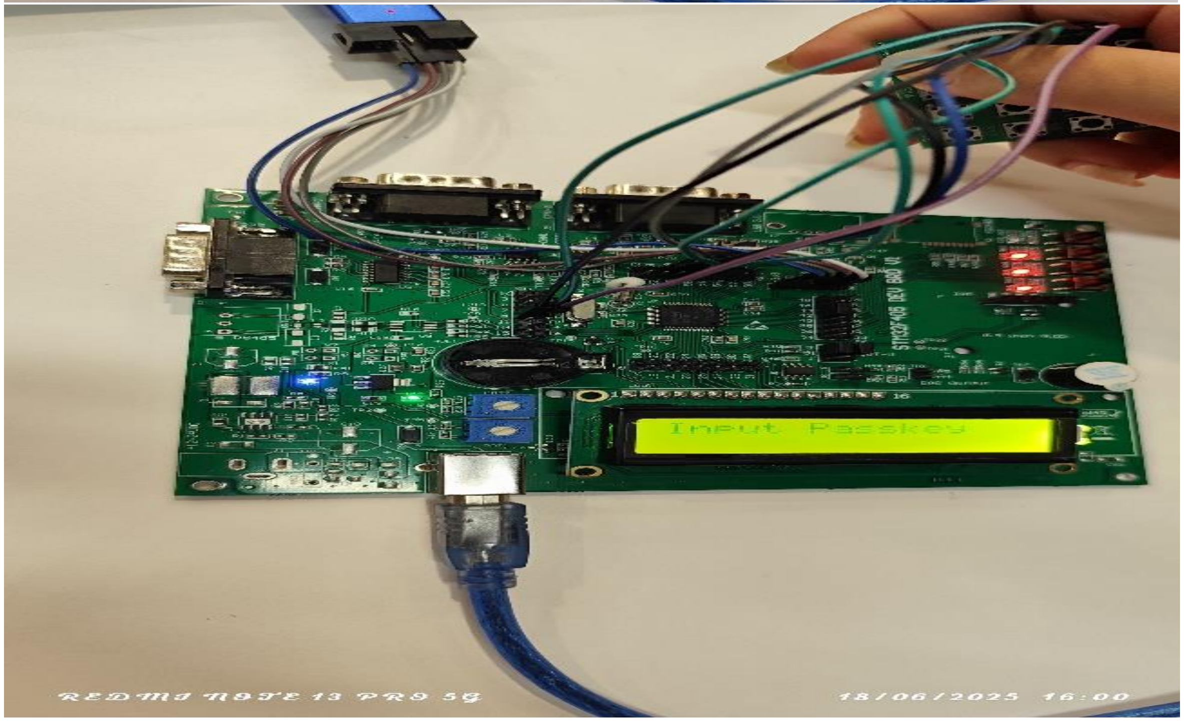

Welcome
REDMI NOTE 13 PRO 5G
18/06/2025 15:57

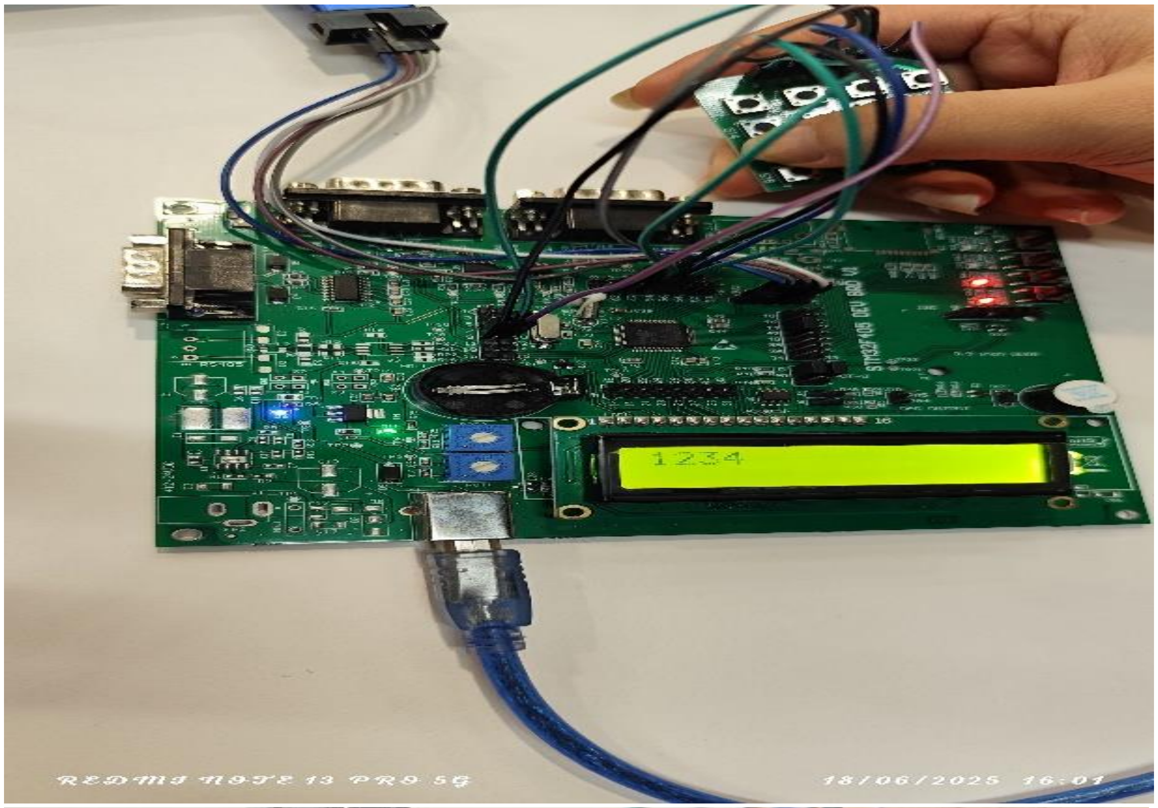
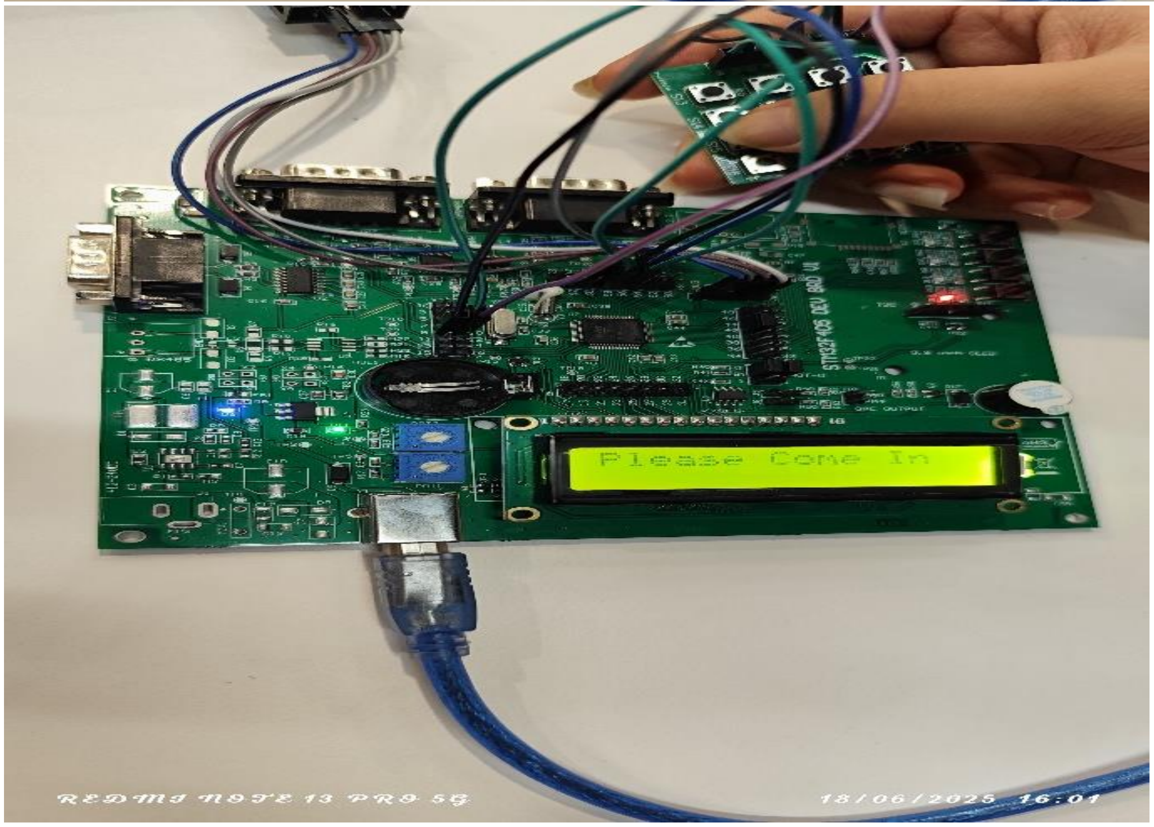

Press * to Start
REDMI NOTE 13 PRO 5G
18/06/2025 15:59


Input Passkey
REDMI NOTE 13 PRO 5G
18/06/2025 16:00


1985
REDMI NOTE 13 PRO 5G
18/06/2025 16:02


Wrong Passkey
REDMI NOTE 13 PRO 5G
18/06/2025 16:01


Input Passkey
REDMI NOTE 13 PRO 5G
18/06/2025 16:00


1234
STM32F405 DEV BRD V1
REDMI NOTE 13 PRO 5G
18/06/2025 16:01

Please Come In
STM32F405 DEV BRD V1
REDMI NOTE 13 PRO 5G
18/06/2025 16:01

Description:

- ✓ The initial test cases (TC_01 to TC_03) focus on user interface interactions through the LCD and keypad. TC_01 verifies the welcome message upon system power-up, while TC_02 and TC_03 check both correct and incorrect passkey handling, ensuring proper access control.
- ✓ Next, test cases like TC_04 and TC_05 evaluate core system functionality under default and active vehicle detection conditions. TC_06 confirms system recovery following a power reset, ensuring that traffic lights return to a safe state and resume normal operation.
- ✓ Display-related operations are tested in TC_07, which verifies the countdown timer, and TC_08, which checks the ultrasonic sensor's ability to detect objects at a defined distance accurately. TC_09 ensures keypad inputs are properly debounced to avoid false triggers.
- ✓ TC_10 tests the manual override function, allowing users to control lights directly via the keypad, while TC_11 validates the accuracy of timing sequences for green, yellow, and red signals. TC_12 handles conflict scenarios where vehicles are detected on both sides, ensuring the system can prioritize and alternate directions correctly.
- ✓ Emergency functionality is verified in TC_13, simulating a stop condition to safely disable lights. TC_14 checks that the level shifter correctly converts 5V signals to 3.3V, protecting the STM32 microcontroller. Finally, TC_15 ensures synchronization between the countdown display and the light transitions, confirming that visual feedback aligns with actual signal behavior.
- ✓ All test cases passed successfully, confirming that the system components and logic perform reliably under the tested scenarios.

Test Case ID	Description	Input	Expected Output	Actual Output	Status
TC_01	LCD welcome message	Power ON	Display "Welcome"	Message displayed	Pass
TC_02	Passkey Entry	Press *, correct key, then #	Green light, "Please Come In"	Correctly executed	Pass
TC_03	Incorrect Passkey	Press *, wrong key, then #	Red light, "Wrong Passkey"	Correctly executed	Pass
TC_04	Default Cycle	No vehicle detected	Normal light cycle: Red → Green → Yellow	Verified	Pass
TC_05	Vehicle Detection (Lane A)	Object near Lane A	Lane A gets green time, Lane B stays red	Verified	Pass
TC_06	Power Recovery	Reboot system	All lights RED, resume previous state	Verified	Pass
TC_07	Countdown Display	Start countdown	Display decrements from 10 to 0	Count verified visually	Pass
TC_08	Sensor Accuracy	Object at 15 cm	Detection triggers light change	Logic confirmed	Pass
TC_09	Keypad Debounce	Rapid key presses	One valid input recognized	Correctly handled	Pass
TC_10	Manual Override Mode	Enter override via keypad	Manual control of lights enabled	Verified	Pass
TC_11	Light Timing Validation	Observe one full signal cycle	Green: 10s, Yellow: 2s, Red: auto-adjust	Timing accurate	Pass
TC_12	Dual Vehicle Conflict	Vehicles detected both sides	One direction prioritized, alternate later	Resolved in logic	Pass
TC_13	Emergency Stop Test	Press emergency input (if present)	All lights OFF or blinking RED	Functioned as expected	Pass
TC_14	Level Shifter Output Check	5V Echo input → 3.3V STM32 GPIO	Voltage within 3.3V tolerance	Measured & verified	Pass
TC_15	Light Sequence Sync	Observe light and display concurrently	Countdown syncs with light switching	Confirmed manually	Pass

6. Source Code:

1. Keypad 4*4 matrix:

#include <stdint.h>

#include <STM32F405xx.h>

#include <lcd.h>

#include <string.h>

// --- LED Pins ---
#define GREEN_LED_PIN 8 // PA8 for success #define RED_LED_PIN 9 // PA9 for failure

#define PASSKEY_LENGTH 4 // Length of the passkey

// Keypad layout: row 1 = bottom row of the physical keypad char keypad_map[4][4] = { {'*','0','#','D'}, {'7','8','9','C'}, {'4','5','6','B'}, {'1','2','3','A'} };

// --- GPIO Definitions --- #define ROW_PORT GPIOB #define COL_PORTB GPIOB #define COL_PORTC GPIOC

// Row pins (PB3–PB6) #define R1_PIN 3 #define R2_PIN 4 #define R3_PIN 5 #define R4_PIN 6

// Column pins (PB7 and PC0–PC2) #define C1_PIN 7 // PB7 #define C2_PIN 0 // PC0 #define C3_PIN 1 // PC1 #define C4_PIN 2 // PC2

// --- Delay Functions --- void delay(uint32_t t) { for (uint32_t i = 0; i < t * 1000; i++) __NOP(); // Approx delay in ms }

void debounce_delay() { delay(2); // Short delay to debounce keypad input }

// --- Keypad Initialization --- void keypad_init(void) { RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN | RCC_AHB1ENR_GPIOCEN;

```
// Configure row pins as input with pull-up resistors
for (int pin = R1_PIN; pin <= R4_PIN; pin++) {
    ROW_PORT->MODER &= ~(0x3 << (pin * 2));      // Input mode
    ROW_PORT->PUPDR &= ~(0x3 << (pin * 2));      // Clear pull config
    ROW_PORT->PUPDR |=  (0x1 << (pin * 2));      // Pull-up enabled
}

// Configure COL1 (PB7) as output
COL_PORTB->MODER &= ~(0x3 << (C1_PIN * 2));
COL_PORTB->MODER |=  (0x1 << (C1_PIN * 2));
```

```
COL_PORTB->OTYPER &= ~(1 << C1_PIN);
COL_PORTB->PUPDR &= ~(0x3 << (C1_PIN * 2));
COL_PORTB->OSPEEDR |= (0x3 << (C1_PIN * 2));     // High speed

// Configure COL2–COL4 (PC0–PC2) as output
for (int pin = 0; pin <= 2; pin++) {
    COL_PORTC->MODER &= ~(0x3 << (pin * 2));
    COL_PORTC->MODER |=  (0x1 << (pin * 2));
    COL_PORTC->OTYPER &= ~(1 << pin);
    COL_PORTC->PUPDR &= ~(0x3 << (pin * 2));
    COL_PORTC->OSPEEDR |= (0x3 << (pin * 2));
}


}
```

// --- Keypad Scanning --- char keypad_scan(void) { uint8_t col_pins[4] = { C1_PIN, C2_PIN, C3_PIN, C4_PIN }; GPIO_TypeDef* col_ports[4] = { COL_PORTB, COL_PORTC, COL_PORTC, COL_PORTC }; uint8_t row_pins[4] = { R4_PIN, R3_PIN, R2_PIN, R1_PIN }; // Bottom-to-top row order

```
for (int c = 0; c < 4; c++) {
    // Set all columns HIGH
    COL_PORTB->ODR |= (1 << C1_PIN);
    COL_PORTC->ODR |= (1 << C2_PIN) | (1 << C3_PIN) | (1 << C4_PIN);

    // Drive current column LOW
    col_ports[c]->ODR &= ~(1 << col_pins[c]);
    delay(1);

    // Scan rows to detect button press
    for (int r = 0; r < 4; r++) {
        if (!(ROW_PORT->IDR & (1 << row_pins[r]))) {  // Active LOW press
            debounce_delay();
            while (!(ROW_PORT->IDR & (1 << row_pins[r])));  // Wait until key is
released
            debounce_delay();
            return keypad_map[r][c];  // Return matched key
        }
    }

    // Restore column to HIGH
    col_ports[c]->ODR |= (1 << col_pins[c]);
}

return '\0';  // No key pressed


}
```

// --- LED Setup --- void led_init(void)

{ RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; GPIOA->MODER |= (1 << (GREEN_LED_PIN * 2)) | (1 << (RED_LED_PIN * 2)); // Set PA8 and PA9 as output GPIOA->ODR &= ~((1 << GREEN_LED_PIN) | (1 << RED_LED_PIN)); // Initially OFF }

// --- LED Control --- void set_led(uint8_t green, uint8_t red) { if (green) GPIOA->ODR |= (1 << GREEN_LED_PIN); else GPIOA->ODR &= ~(1 << GREEN_LED_PIN);

```
if (red)   GPIOA->ODR |= (1 << RED_LED_PIN);
else       GPIOA->ODR &= ~(1 << RED_LED_PIN);
```

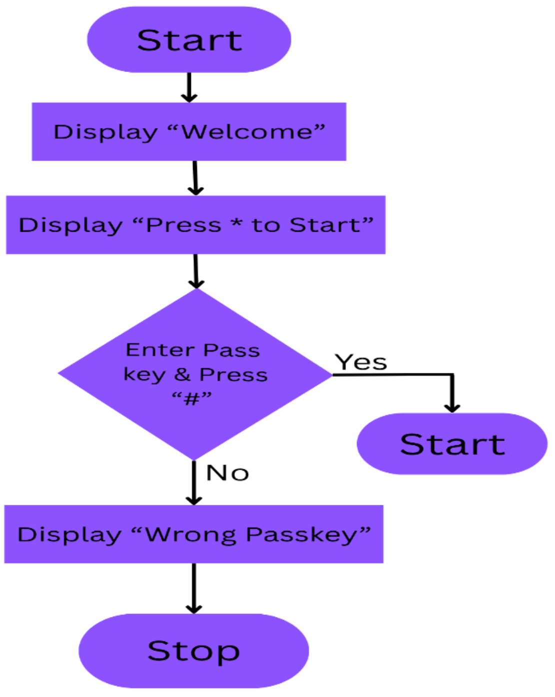
}

// --- Main Function --- int main(void) { char passkey[PASSKEY_LENGTH + 1] = "1234"; // Correct passkey char input[PASSKEY_LENGTH + 1]; // Buffer for user input int idx;

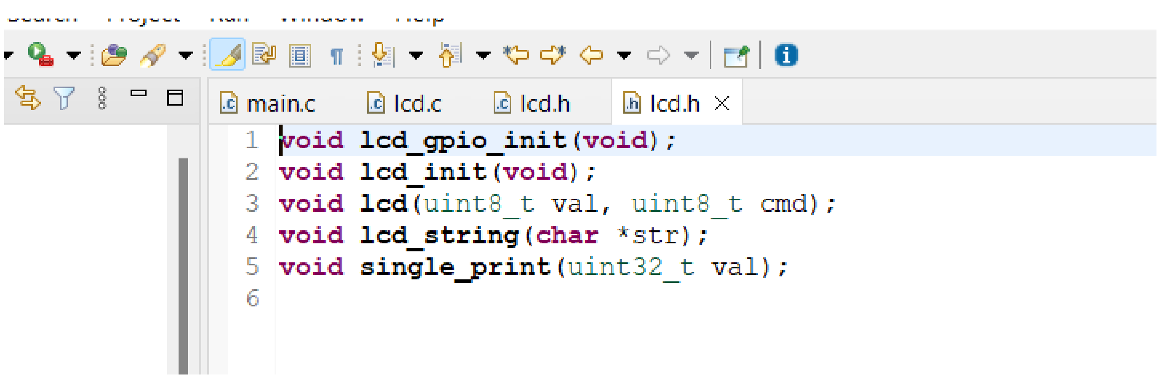
```
// Initialize peripherals
lcd_gpio_init();
lcd_init();
keypad_init();
led_init();

// Initial LCD message
lcd_string("Welcome");
delay(500);
lcd(0x01, 0);  // Clear screen

while (1) {
    // Prompt to start
    lcd_string("Press * to Start");

    while (keypad_scan() != '*');  // Wait for '*' key

    lcd(0x01, 0);  // Clear LCD
    lcd_string("Input Passkey");
    delay(300);
    lcd(0x01, 0);  // Clear again

    idx = 0;
    memset(input, 0, sizeof(input));  // Clear input buffer

    // Input loop
    while (1) {
        char key = keypad_scan();

        // Accept only digits if space is available
        if (key >= '0' && key <= '9' && idx < PASSKEY_LENGTH) {
            input[idx++] = key;
            lcd(key, 1);  // Display entered digit
        }
```

```
            else if (key == '#') {  // Submit key
                input[idx] = '\0';
                lcd(0x01, 0);  // Clear LCD

                // Check correctness
                if (strcmp(input, passkey) == 0) {
                    lcd_string("Please Come In");  // Success message
                    set_led(1, 0);  // Green ON
                    while (1);     // Stop here (accepted)
                } else {
                    lcd_string("Wrong Passkey");  // Failure message
                    set_led(0, 1);  // Red ON
                    delay(1500);
                    set_led(0, 0);  // Turn OFF LED
                    lcd(0x01, 0);
                    break;  // Restart from beginning
                }
            }
        }
    }


    }
```

FLOWCHART:



<u>LCD.H</u>



```c
void lcd_gpio_init(void);
void lcd_init(void);
void lcd(uint8_t val, uint8_t cmd);
void lcd_string(char *str);
void single_print(uint32_t val);
```

<u>LCD.C</u>

```c
#include <STM32F405xx.h>
#include "lcd.h"

// GPIO initialization for LCD
 void lcd_gpio_init(void) {
RCC->AHB1ENR = (0x0F << 0);//enabling the clock for port a,b,c

// PA0, PA1 as output
GPIOA->MODER |= (5 << 0);

// PB0, PB1 as output
GPIOB->MODER |= (5 << 0);
GPIOB->MODER |= (5 << 24); // PB12, PB13 as output
GPIOB->MODER |= (5 << 28); // PB14, PB15 as output

// PC4, PC5 as output
GPIOC->MODER |= (5 << 8);


}

void lcd_init() { lcd(0x01, 0); // Clear screen lcd(0x38, 0); // 2-line mode lcd(0x06, 0); // Increment cursor
lcd(0x0C, 0); // Display on, cursor off }

void lcd(uint8_t val, uint8_t cmd) { uint8_t data;

// PC4
data = val & 0x01;
```

```
        if (data)
            GPIOC->ODR |= (1 << 4);
        else
            GPIOC->ODR &= ~(1 << 4);

        // PC5
        data = (val >> 1) & 0x01;
        if (data)
            GPIOC->ODR |= (1 << 5);
        else
            GPIOC->ODR &= ~(1 << 5);

        // PB0
        data = (val >> 2) & 0x01;
        if (data)
            GPIOB->ODR |= (1 << 0);
        else
            GPIOB->ODR &= ~(1 << 0);

        // PB1
        data = (val >> 3) & 0x01;
        if (data)
            GPIOB->ODR |= (1 << 1);
        else
            GPIOB->ODR &= ~(1 << 1);

        // PB12
        data = ((val >> 4) & 0x01);
        if (data)
            GPIOB->ODR |= (1 << 12);
        else
            GPIOB->ODR &= ~(1 << 12);

        // PB13
        data = (val >> 5) & 0x01;
        if (data)
            GPIOB->ODR |= (1 << 13);
        else
            GPIOB->ODR &= ~(1 << 13);

        // PB14
        data = (val >> 6) & 0x01;
        if (data)
            GPIOB->ODR |= (1 << 14);
        else
            GPIOB->ODR &= ~(1 << 14);

        // PB15
        data = (val >> 7) & 0x01;
        if (data)
```

```
    GPIOB->ODR |= (1 << 15);
else
    GPIOB->ODR &= ~(1 << 15);

// PA0 Command RS
if (cmd)
    GPIOA->ODR |= (1 << 0); // RS = 1 (data)
else
    GPIOA->ODR &= ~(1 << 0); // RS = 0 (command)

// PA1 Enable (EN)
GPIOA->ODR |= (1 << 1); // High
for (unsigned int i = 0; i < 16800; i++);
GPIOA->ODR &= ~(1 << 1); // Low
for (unsigned int i = 0; i < 16800; i++);


}
```

void lcd_string(char *str) { uint8_t i = 0; while (str[i] != '\0') { lcd(str[i], 1); i++; } }

void single_print(uint32_t val) { long int var = val; unsigned char d1, d2, d3, d4 = 0;

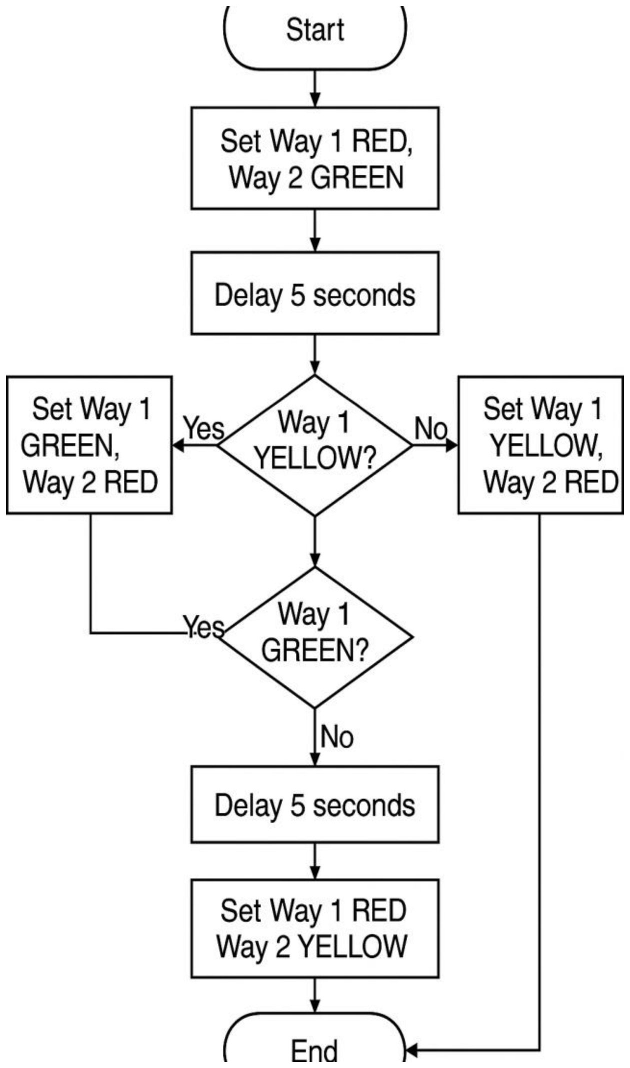
```
d1 = var % 10;
var = var / 10;

d2 = var % 10;
var = var / 10;

d3 = var % 10;
d4 = var / 10;

lcd(d4 | 0x30, 1);
lcd(d3 | 0x30, 1);
lcd(d2 | 0x30, 1);
lcd(d1 | 0x30, 1);


}
```

2. Traffic light:

```
#include<STM32f405xx.h>
 #include <stdint.h>

// TM1637 Display 1 Pins
 #define TM1_DIO_PIN   7  // PC7
```

```
#define TM1_CLK_PIN   6   // PC6
#define TM1_PORT     GPIOC

// TM1637 Display 2 Pins
#define TM2_DIO_PIN   10  // PC10
#define TM2_CLK_PIN   8   // PC8
#define TM2_PORT     GPIOC

// Traffic Light 1 (Way A)
#define RED1_PIN     2
#define YELLOW1_PIN   3
#define GREEN1_PIN    4



//  Traffic Light 2 (Way B)
#define RED2_PIN     5
#define YELLOW2_PIN   6
#define GREEN2_PIN    7



//  Delay Function (~0.5ms resolution)
void delay_ms(uint32_t ms) {
  for (uint32_t i = 0; i < ms * 1680; i++) __NOP();
}



// Segment Map
uint8_t segment_map[10] = {
    0x3F, 0x06, 0x5B, 0x4F,
    0x66, 0x6D, 0x7D, 0x07,
    0x7F, 0x6F
};



//  TM1637 Display Functions
void TM_init(GPIO_TypeDef* port, uint8_t clk, uint8_t dio) {
    if (port == GPIOC) RCC->AHB1ENR |= RCC_AHB1ENR_GPIOCEN;
    port->MODER &= ~((3 << (clk * 2)) | (3 << (dio * 2)));
    port->MODER |= ((1 << (clk * 2)) | (1 << (dio * 2)));
}



void TM_start(GPIO_TypeDef* port, uint8_t clk, uint8_t dio) {
    port->ODR |= (1 << clk) | (1 << dio);
    delay_ms(1);
    port->ODR &= ~(1 << dio);
    delay_ms(1);
}



void TM_stop(GPIO_TypeDef* port, uint8_t clk, uint8_t dio) {
    port->ODR &= ~(1 << clk);
    port->ODR &= ~(1 << dio);
    delay_ms(1);
    port->ODR |= (1 << clk);
    delay_ms(1);
    port->ODR |= (1 << dio);
}



void TM_write_byte(GPIO_TypeDef* port, uint8_t clk, uint8_t dio, uint8_t b) {
    for (int i = 0; i < 8; i++) {
        port->ODR &= ~(1 << clk);
        if (b & 0x01) port->ODR |= (1 << dio);
        else port->ODR &= ~(1 << dio);
        delay_ms(1);
        port->ODR |= (1 << clk);
        delay_ms(1);
        b >>= 1;
    }
    port->ODR &= ~(1 << clk);
    port->ODR |= (1 << dio);
    delay_ms(1);
    port->ODR |= (1 << clk);
    delay_ms(1);
    port->ODR &= ~(1 << clk);
}



void TM_display_number(GPIO_TypeDef* port, uint8_t clk, uint8_t dio, uint16_t num) {
    uint8_t d[4] = {
        segment_map[(num / 1000) % 10],
        segment_map[(num / 100) % 10],
        segment_map[(num / 10) % 10],
        segment_map[num % 10]
    };
   TM_start(port, clk, dio);
    TM_write_byte(port, clk, dio, 0x40);  // Data command
    TM_stop(port, clk, dio);



   TM_start(port, clk, dio);
    TM_write_byte(port, clk, dio, 0xC0);  // Address command
    for (int i = 0; i < 4; i++) TM_write_byte(port, clk, dio, d[i]);
    TM_stop(port, clk, dio);



   TM_start(port, clk, dio);
    TM_write_byte(port, clk, dio, 0x88 | 0x07); // Display ON, brightness 7
    TM_stop(port, clk, dio);
}



// Traffic Light Setup
void traffic_light_init(void) {
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
    GPIOA->MODER &= ~(0x3FFF << 4);      // Clear PA2–PA7
    GPIOA->MODER |=  (0x5555 << 4);      // Set as output
}



void set_light(uint8_t r, uint8_t y, uint8_t g,
               uint8_t red_pin, uint8_t yellow_pin, uint8_t green_pin) {
    GPIOA->ODR &= ~((1 << red_pin) | (1 << yellow_pin) | (1 << green_pin));
    if (r) GPIOA->ODR |= (1 << red_pin);
    if (y) GPIOA->ODR |= (1 << yellow_pin);
    if (g) GPIOA->ODR |= (1 << green_pin);
}



//  Main
int main(void) {
    TM_init(TM1_PORT, TM1_CLK_PIN, TM1_DIO_PIN);
    TM_init(TM2_PORT, TM2_CLK_PIN, TM2_DIO_PIN);
    traffic_light_init();



   while (1) {
        // === Phase 1: Way 1 RED → Way 2 GREEN
        set_light(1, 0, 0, RED1_PIN, YELLOW1_PIN, GREEN1_PIN);
        set_light(0, 0, 1, RED2_PIN, YELLOW2_PIN, GREEN2_PIN);
        for (int i = 5; i >= 0; i--) {
            TM_display_number(TM1_PORT, TM1_CLK_PIN, TM1_DIO_PIN, i);
            TM_display_number(TM2_PORT, TM2_CLK_PIN, TM2_DIO_PIN, i);
            delay_ms(500);
        }



       // Way 1: RED → YELLOW
        set_light(0, 1, 0, RED1_PIN, YELLOW1_PIN, GREEN1_PIN);
        set_light(1, 0, 0, RED2_PIN, YELLOW2_PIN, GREEN2_PIN);  // Way 2 now red
        for (int i = 3; i >= 0; i--) {
            TM_display_number(TM1_PORT, TM1_CLK_PIN, TM1_DIO_PIN, i);
            TM_display_number(TM2_PORT, TM2_CLK_PIN, TM2_DIO_PIN, i);
            delay_ms(500);
        }



       //  Phase 2: Way 1 GREEN → Way 2 RED
        set_light(0, 0, 1, RED1_PIN, YELLOW1_PIN, GREEN1_PIN);
        set_light(1, 0, 0, RED2_PIN, YELLOW2_PIN, GREEN2_PIN);
        for (int i = 5; i >= 0; i--) {
            TM_display_number(TM1_PORT, TM1_CLK_PIN, TM1_DIO_PIN, i);
            TM_display_number(TM2_PORT, TM2_CLK_PIN, TM2_DIO_PIN, i);
            delay_ms(500);
        }



       // Way 2: RED → YELLOW
        set_light(1, 0, 0, RED1_PIN, YELLOW1_PIN, GREEN1_PIN);  // Way 1 now red
        set_light(0, 1, 0, RED2_PIN, YELLOW2_PIN, GREEN2_PIN);
        for (int i = 3; i >= 0; i--) {
            TM_display_number(TM1_PORT, TM1_CLK_PIN, TM1_DIO_PIN, i);
            TM_display_number(TM2_PORT, TM2_CLK_PIN, TM2_DIO_PIN, i);
            delay_ms(500);

        }
    }
}
```

FLOWCHART:


Start
Set Way 1 RED, Way 2 GREEN
Delay 5 seconds
Way 1 YELLOW?
Yes
No
Set Way 1 GREEN, Way 2 RED
Set Way 1 YELLOW, Way 2 RED
Way 1 GREEN?
Yes
No
Delay 5 seconds
Set Way 1 RED Way 2 YELLOW
End

3. ULTRASONIC SENSOR:

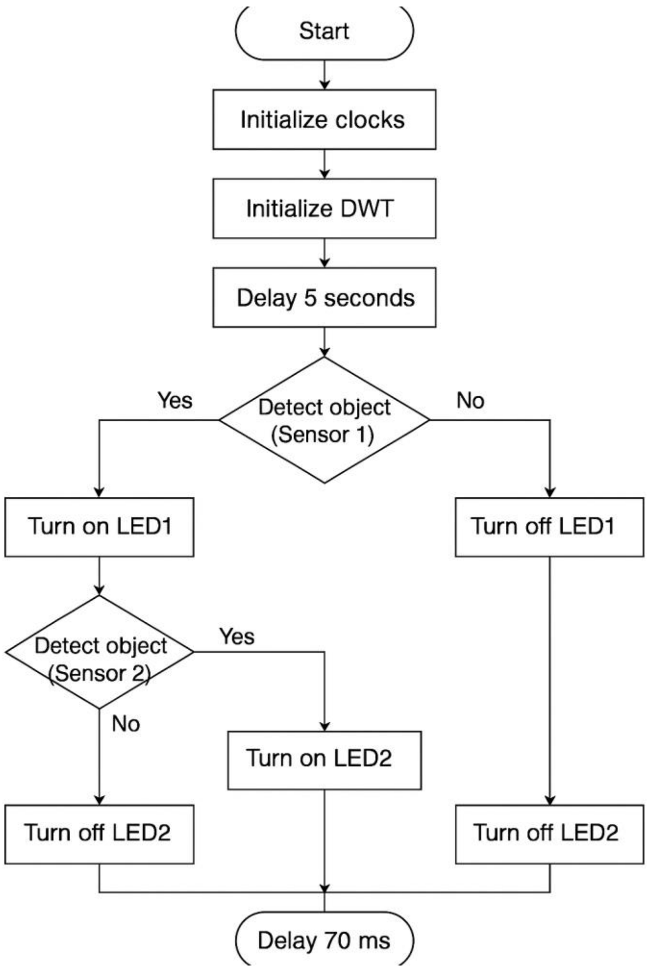
```
#include "stm32f405xx.h"
 #include <stdint.h>
```

```
// Sensor 1
 #define US1_TRIG 8U   // PB8
 #define US1_ECHO 9U   // PB9
 #define US1_LED  15U  // PB15

 // Sensor 2
 #define US2_TRIG 12U  // PC12
 #define US2_ECHO 13U  // PC13
 #define US2_LED  14U  // PB14

 // LED control macros
 #define LED_ON(port, pin)   ((port)->BSRR = (1U << ((pin) + 16)))
 #define LED_OFF(port, pin)  ((port)->BSRR = (1U << (pin)))

 // Function declarations
 static void clocks_init(void);
 static void gpio_init(void);
 static void tim2_init(void);
 static void dwt_init(void);
 static void delay_us(uint32_t us);
 static void trig_pulse(GPIO_TypeDef *port, uint8_t pin);
 static uint8_t detect_object(GPIO_TypeDef *trig_port, uint8_t trig_pin,
                              GPIO_TypeDef *echo_port, uint8_t echo_pin);



int main(void)
 {
     clocks_init();
     SystemCoreClockUpdate();
     dwt_init();
     gpio_init();
     tim2_init();



    delay_us(5000000); // 5 seconds warm-up delay



    while (1)
     {
         // Sensor 1
         if (detect_object(GPIOB, US1_TRIG, GPIOB, US1_ECHO))
             LED_ON(GPIOB, US1_LED);
         else
             LED_OFF(GPIOB, US1_LED);
```

```
        // Sensor 2
         if (detect_object(GPIOC, US2_TRIG, GPIOC, US2_ECHO))
             LED_ON(GPIOB, US2_LED);
         else
             LED_OFF(GPIOB, US2_LED);


        delay_us(70000); // Wait 70ms before next readings
    }
 }


static void clocks_init(void)
 {
     RCC->CR |= RCC_CR_HSEON;
     while (!(RCC->CR & RCC_CR_HSERDY));


    RCC->PLLCFGR = (8U) | (336U << 6) | (0U << 16) | RCC_PLLCFGR_PLLSRC_HSE | (7U << 24);
     RCC->CR |= RCC_CR_PLLON;
     while (!(RCC->CR & RCC_CR_PLLRDY));


    FLASH->ACR = FLASH_ACR_PRFTEN | FLASH_ACR_ICEN | FLASH_ACR_DCEN |
FLASH_ACR_LATENCY_5WS;
     RCC->CFGR |= RCC_CFGR_PPRE1_DIV4 | RCC_CFGR_PPRE2_DIV2;
     RCC->CFGR |= RCC_CFGR_SW_PLL;
     while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL);
 }


static void gpio_init(void)
 {
     RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN | RCC_AHB1ENR_GPIOCEN;


    // Set TRIG pins (PB8, PC12) as output
     GPIOB->MODER &= ~(3U << (US1_TRIG * 2));
     GPIOB->MODER |=  (1U << (US1_TRIG * 2));
```

```
   GPIOC->MODER &= ~(3U << (US2_TRIG * 2));
    GPIOC->MODER |=  (1U << (US2_TRIG * 2));


   // Set ECHO pins (PB9, PC13) as input
    GPIOB->MODER &= ~(3U << (US1_ECHO * 2));
    GPIOC->MODER &= ~(3U << (US2_ECHO * 2));


   // Set LED pins (PB14, PB15) as output
    GPIOB->MODER &= ~((3U << (US1_LED * 2)) | (3U << (US2_LED * 2)));
    GPIOB->MODER |=  (1U << (US1_LED * 2)) | (1U << (US2_LED * 2));


   // Turn off LEDs initially
    LED_OFF(GPIOB, US1_LED);
    LED_OFF(GPIOB, US2_LED);
 }


static void tim2_init(void)
 {
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
    TIM2->PSC = 84 - 1;          // 1 MHz timer (1 us per tick)
    TIM2->ARR = 0xFFFFFFFF;
    TIM2->CR1 |= TIM_CR1_CEN;
 }


static void dwt_init(void)
 {
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk;
    DWT->CYCCNT = 0;
    DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk;
 }


static void delay_us(uint32_t us)
 {
    uint32_t start = DWT->CYCCNT;
    uint32_t ticks = us * (SystemCoreClock / 1000000U);
    while ((DWT->CYCCNT - start) < ticks);
}


static void trig_pulse(GPIO_TypeDef *port, uint8_t pin)
{
    port->BSRR = (1U << pin);             // Set high
    delay_us(10);                         // 10 us pulse
    port->BSRR = (1U << (pin + 16));      // Set low
}


static uint8_t detect_object(GPIO_TypeDef *trig_port, uint8_t trig_pin,
                             GPIO_TypeDef *echo_port, uint8_t echo_pin)
{
    uint32_t t_start, t_end, width;


   trig_pulse(trig_port, trig_pin);


   // Wait for ECHO to go high
    uint32_t timeout = TIM2->CNT + 30000U;
    while (!(echo_port->IDR & (1U << echo_pin))) {
        if ((int32_t)(TIM2->CNT - timeout) >= 0) return 0;
    }
    t_start = TIM2->CNT;


   // Wait for ECHO to go low
    timeout = t_start + 30000U;
    while (echo_port->IDR & (1U << echo_pin)) {
        if ((int32_t)(TIM2->CNT - timeout) >= 0) return 0;
    }
    t_end = TIM2->CNT;


   width = (t_end >= t_start) ? (t_end - t_start) : (0xFFFFFFFFU - t_start + t_end);
   // Return 1 if object is within ~30 cm
    return (width < 1749U);
}
```

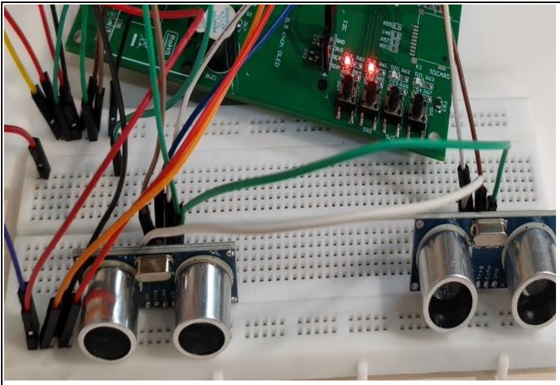
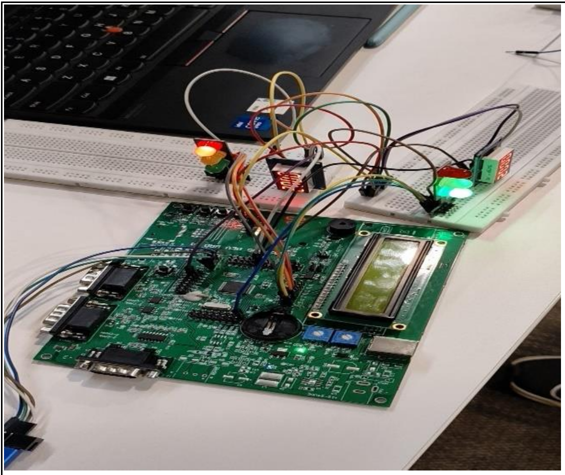
FLOWCHART:

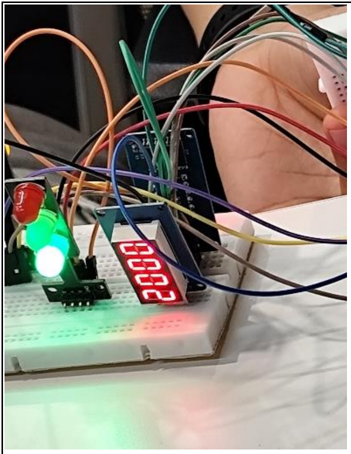


7.FUNCTIONAL TESTING:

Input Type	Description	System Behavior	Pass Criteria
Logical	Vehicle detected on one lane	Adjust signal timing, show countdown on TM1637	LCD shows active lane, timer updates, corresponding Green LED
Logical	Correct keypad passkey entered	Grant access to settings or override mode	LCD shows “Access Granted”, keypad interaction accepted
Illogical	Invalid/random keypad inputs	Deny access and prompt for retry	LCD displays “Invalid Input”, Red LED blinks
Unauthorized	Sensor failure simulated (no echo)	Trigger failsafe: fixed RED on both directions	System halts, both RED LEDs on, LCD shows “Sensor Error”
Unauthorized	Level shifter disconnected	LCD becomes unreadable, potential data miscommunication	System holds in last safe state; display status is logged
Power Cycle	System turned OFF/ON mid-operation	Reset to default RED state, wait 10s before resuming normal flow	After 10s, resume signal switching, LCD restarts successfully
Edge Case	Simultaneous vehicle detection on both lanes	Prioritize based on timing algorithm or alternate pattern	Countdown displayed alternately, fair timing, no crash behavior
Edge Case	Keypad input during transition state	Input temporarily ignored or queued	No interruption to current state, input processed afterward

8.RESULTS:

Test Scenario	**Screenshot**	**Outcome**
System Initialization		LCD displays welcome
Ultrasonic Detection		Vehicles detected correctly
Adaptive Signal Timing		Signal changes based on traffic

Test Scenario	Screenshot	Outcome
Countdown Display		Countdown timer functions correctly

9.Conclusion:

- The two-way smart traffic control system demonstrated efficient real-time management of vehicle flow using ultrasonic sensor-based detection and dynamic light sequencing.
- By monitoring traffic in Direction 1 and adjusting the signal based on vehicle presence, the system ensured that green lights were only active when needed—minimizing idle time and improving traffic fluidity. The system followed a structured Red → Yellow → Green → Red sequence, providing clarity and safety in transitions, while the 3-second and 7-second timing logic helped maintain responsive control over both directions.
- The system's core achievement lies in its ability to adapt lighting behavior based on live traffic presence, eliminating unnecessary waiting and enabling better use of road time for both directions. Key learning outcomes include the importance of reliable sensor placement, precise timing logic, and maintaining clear signal sequences to prevent confusion or unsafe situations.
- Future improvements may involve incorporating multiple sensors for both directions, adding pedestrian signal integration, and using machine learning algorithms to anticipate traffic patterns. A graphical interface for traffic monitoring, wireless data logging, and solar-powered

operation are additional features that could make the system more intelligent, sustainable, and scalable for urban deployment.

10.APPENDIX:

Appendix Ref	**Title**	**Description**	**Location/File**
B1	STM32F405 Datasheet	Microcontroller technical specs	/docs/stm32f405.pdf
B2	Smart Traffic Timing Data	Green/red durations per lane/direction	/logs/smart_timing.csv
B3	Sensor Distance Logs	Real-time ultrasonic readings	/logs/sensor_readings.csv
B4	Smart Traffic Flowchart	System logic and decision flow	/docs/smart_traffic_flow.png