NORTH SOUTH UNIVERSITY

Department of Electrical & Computer Engineering

FINAL REPORT SUBMISSION

KEYPAD SYSTEM

Group Members:

	Name	ID	NSU Email
1	Samiur Rahman Alif	2021088642	samiur.alif@northsouth.edu
2	HM Ashiqur Rahman	2013941642	ashiqur.rahman123@northsouth.edu
3	Md Sadikur Rahman Sadeed	2112251642	sadikur.sadeed@northsouth.edu
4	Imtiaz Ahmed	2013552642	Imtiaz.ahmed@northsouth.edu

Faculty: Md. Arifur Rahman

Project Advisor: Asif Haider Rafi

Department of Electrical & Computer Engineering (ECE)

ABSTRACT

The **Keypad System** is a simple project featuring a 4x4 keypad matrix interfaced with an STM32F103C8T6 – Blue Pill microcontroller. By configuring the GPIO pins, using a multiplexing technique, the system scans for key presses based on the row and column combinations. The code is uploaded via ST-Link V2 (debugger / programmer), and the overall system captures keypad inputs for applications like password entry or device control, ensuring reliable user interaction with efficient scanning and debouncing techniques.

COMPONENTS

1. STM32F103C8T6 (Blue Pill).
2. ST - LINK V2 Debugger / Programmer.
3. 4x4 Keypad Matrix.
4. 3.3V power source via USB.

KEYPAD INTERFACE WITH STM32

- **Wiring:** we connect the 4x4 keypad to the STM32 microcontroller using GPIO pins (A8 to B4). The ST-Link V2 interface is used to upload the code to the STM32, while a USB connection supplies a stable 3.3V power source, ensuring seamless operation and reliable performance.

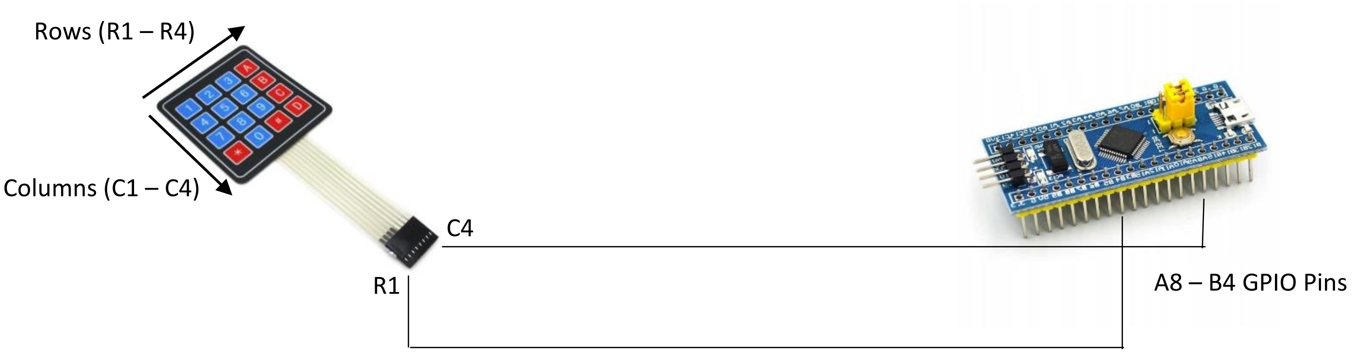


Figure 1: wiring between keypad with STM32

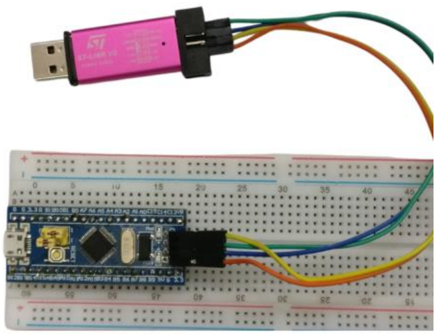
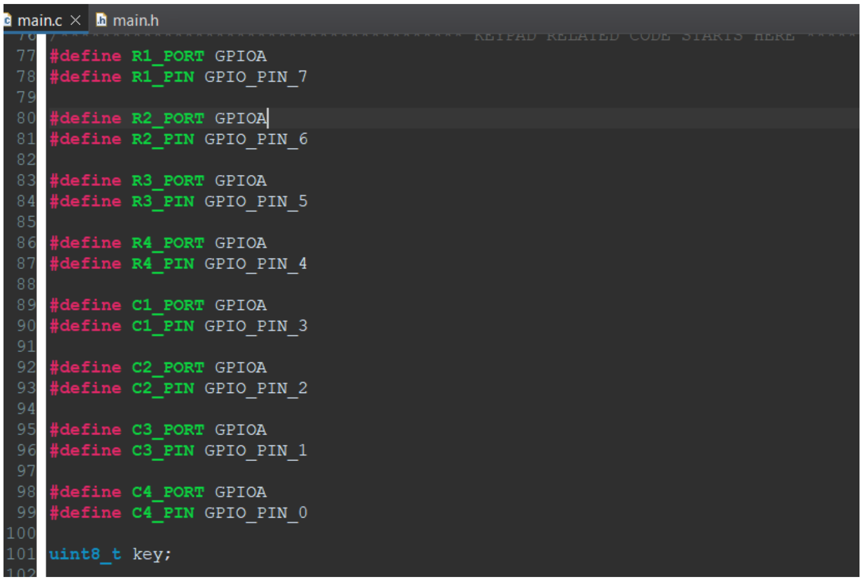
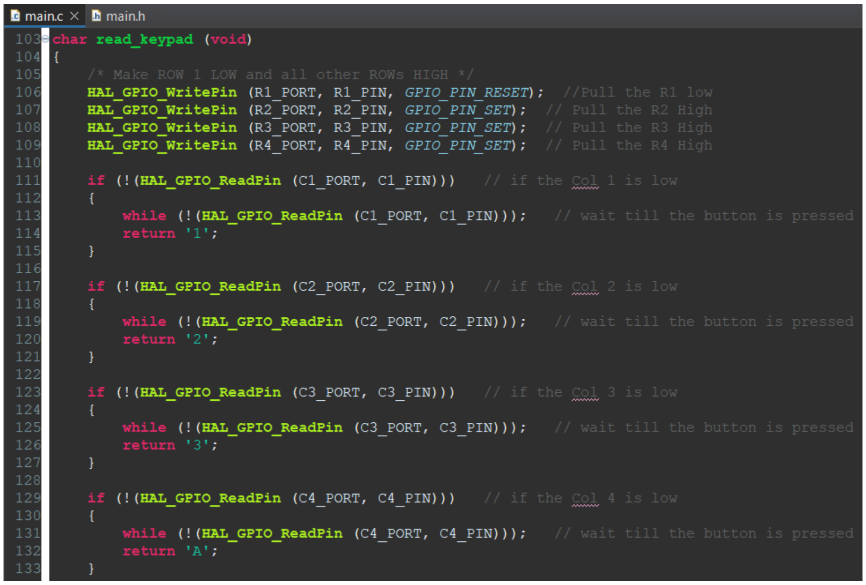


Figure 2: Wiring between ST-Link V2 with STM32

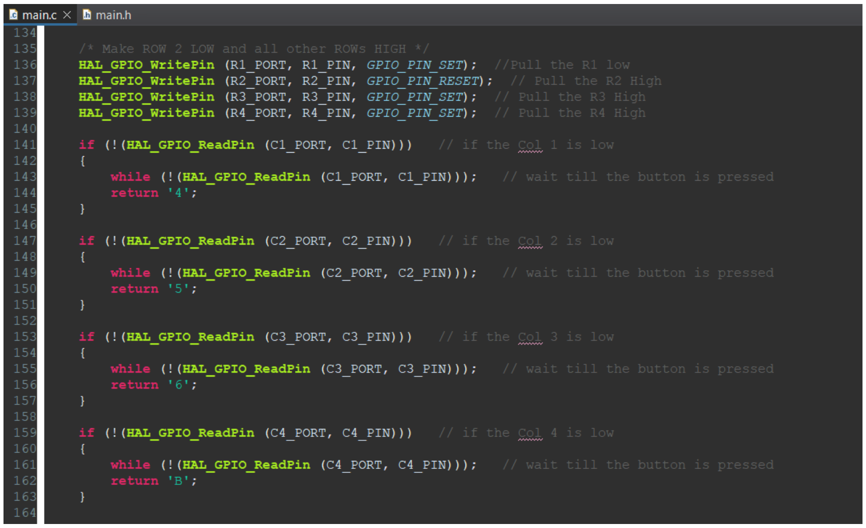
- **Code:** In the coding section, the STM32 scans the 4x4 keypad matrix by configuring GPIO pins for rows and columns. A multiplexing technique is employed to detect key presses efficiently. Debouncing ensures accurate input detection.



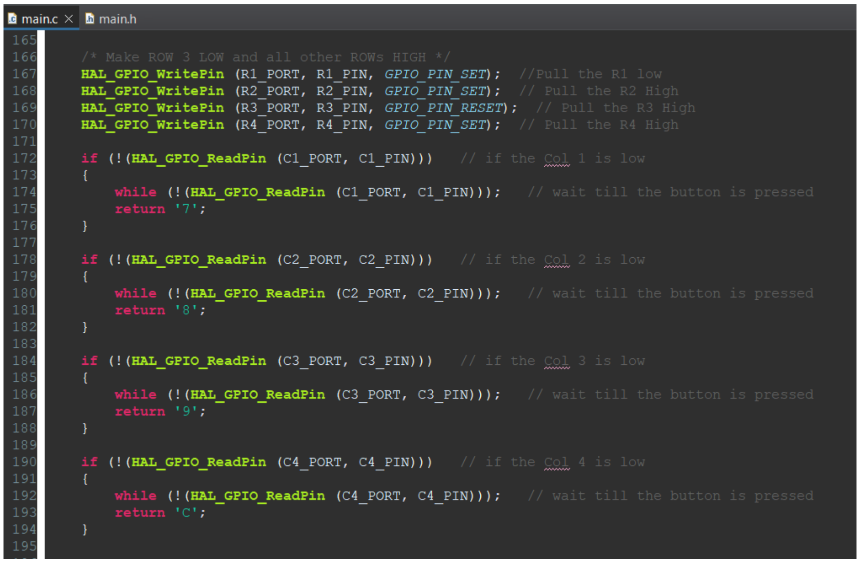
```c
/********************************** KEYPAD RELATED CODE STARTS HERE *****
#define R1_PORT GPIOA
#define R1_PIN GPIO_PIN_7

#define R2_PORT GPIOA
#define R2_PIN GPIO_PIN_6

#define R3_PORT GPIOA
#define R3_PIN GPIO_PIN_5

#define R4_PORT GPIOA
#define R4_PIN GPIO_PIN_4

#define C1_PORT GPIOA
#define C1_PIN GPIO_PIN_3

#define C2_PORT GPIOA
#define C2_PIN GPIO_PIN_2

#define C3_PORT GPIOA
#define C3_PIN GPIO_PIN_1

#define C4_PORT GPIOA
#define C4_PIN GPIO_PIN_0

uint8_t key;
```



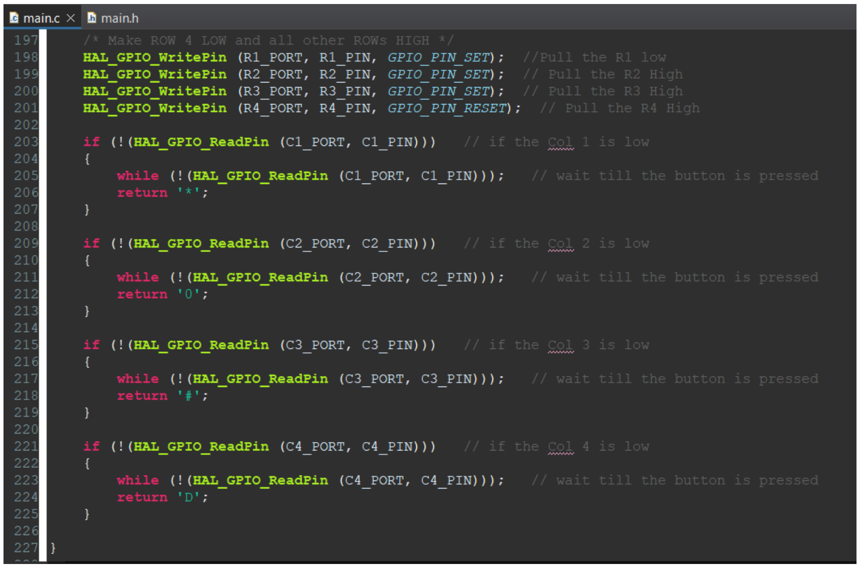
```c
char read_keypad (void)
{
    /* Make ROW 1 LOW and all other ROWs HIGH */
    HAL_GPIO_WritePin (R1_PORT, R1_PIN, GPIO_PIN_RESET);  //Pull the R1 low
    HAL_GPIO_WritePin (R2_PORT, R2_PIN, GPIO_PIN_SET);  // Pull the R2 High
    HAL_GPIO_WritePin (R3_PORT, R3_PIN, GPIO_PIN_SET);  // Pull the R3 High
    HAL_GPIO_WritePin (R4_PORT, R4_PIN, GPIO_PIN_SET);  // Pull the R4 High

    if (!(HAL_GPIO_ReadPin (C1_PORT, C1_PIN)))   // if the Col 1 is low
    {
        while (!(HAL_GPIO_ReadPin (C1_PORT, C1_PIN)));   // wait till the button is pressed
        return '1';
    }

    if (!(HAL_GPIO_ReadPin (C2_PORT, C2_PIN)))   // if the Col 2 is low
    {
        while (!(HAL_GPIO_ReadPin (C2_PORT, C2_PIN)));   // wait till the button is pressed
        return '2';
    }

    if (!(HAL_GPIO_ReadPin (C3_PORT, C3_PIN)))   // if the Col 3 is low
    {
        while (!(HAL_GPIO_ReadPin (C3_PORT, C3_PIN)));   // wait till the button is pressed
        return '3';
    }

    if (!(HAL_GPIO_ReadPin (C4_PORT, C4_PIN)))   // if the Col 4 is low
    {
        while (!(HAL_GPIO_ReadPin (C4_PORT, C4_PIN)));   // wait till the button is pressed
        return 'A';
    }
```



```c

    /* Make ROW 2 LOW and all other ROWs HIGH */
    HAL_GPIO_WritePin (R1_PORT, R1_PIN, GPIO_PIN_SET);  //Pull the R1 low
    HAL_GPIO_WritePin (R2_PORT, R2_PIN, GPIO_PIN_RESET);  // Pull the R2 High
    HAL_GPIO_WritePin (R3_PORT, R3_PIN, GPIO_PIN_SET);  // Pull the R3 High
    HAL_GPIO_WritePin (R4_PORT, R4_PIN, GPIO_PIN_SET);  // Pull the R4 High

    if (!(HAL_GPIO_ReadPin (C1_PORT, C1_PIN)))   // if the Col 1 is low
    {
        while (!(HAL_GPIO_ReadPin (C1_PORT, C1_PIN)));   // wait till the button is pressed
        return '4';
    }

    if (!(HAL_GPIO_ReadPin (C2_PORT, C2_PIN)))   // if the Col 2 is low
    {
        while (!(HAL_GPIO_ReadPin (C2_PORT, C2_PIN)));   // wait till the button is pressed
        return '5';
    }

    if (!(HAL_GPIO_ReadPin (C3_PORT, C3_PIN)))   // if the Col 3 is low
    {
        while (!(HAL_GPIO_ReadPin (C3_PORT, C3_PIN)));   // wait till the button is pressed
        return '6';
    }

    if (!(HAL_GPIO_ReadPin (C4_PORT, C4_PIN)))   // if the Col 4 is low
    {
        while (!(HAL_GPIO_ReadPin (C4_PORT, C4_PIN)));   // wait till the button is pressed
        return 'B';
    }
```



```c

    /* Make ROW 3 LOW and all other ROWs HIGH */
    HAL_GPIO_WritePin (R1_PORT, R1_PIN, GPIO_PIN_SET);  //Pull the R1 low
    HAL_GPIO_WritePin (R2_PORT, R2_PIN, GPIO_PIN_SET);  // Pull the R2 High
    HAL_GPIO_WritePin (R3_PORT, R3_PIN, GPIO_PIN_RESET);  // Pull the R3 High
    HAL_GPIO_WritePin (R4_PORT, R4_PIN, GPIO_PIN_SET);  // Pull the R4 High

    if (!(HAL_GPIO_ReadPin (C1_PORT, C1_PIN)))   // if the Col 1 is low
    {
        while (!(HAL_GPIO_ReadPin (C1_PORT, C1_PIN)));   // wait till the button is pressed
        return '7';
    }

    if (!(HAL_GPIO_ReadPin (C2_PORT, C2_PIN)))   // if the Col 2 is low
    {
        while (!(HAL_GPIO_ReadPin (C2_PORT, C2_PIN)));   // wait till the button is pressed
        return '8';
    }

    if (!(HAL_GPIO_ReadPin (C3_PORT, C3_PIN)))   // if the Col 3 is low
    {
        while (!(HAL_GPIO_ReadPin (C3_PORT, C3_PIN)));   // wait till the button is pressed
        return '9';
    }

    if (!(HAL_GPIO_ReadPin (C4_PORT, C4_PIN)))   // if the Col 4 is low
    {
        while (!(HAL_GPIO_ReadPin (C4_PORT, C4_PIN)));   // wait till the button is pressed
        return 'C';
    }

```



```c
    /* Make ROW 4 LOW and all other ROWs HIGH */
    HAL_GPIO_WritePin (R1_PORT, R1_PIN, GPIO_PIN_SET);  //Pull the R1 low
    HAL_GPIO_WritePin (R2_PORT, R2_PIN, GPIO_PIN_SET);  // Pull the R2 High
    HAL_GPIO_WritePin (R3_PORT, R3_PIN, GPIO_PIN_SET);  // Pull the R3 High
    HAL_GPIO_WritePin (R4_PORT, R4_PIN, GPIO_PIN_RESET);  // Pull the R4 High

    if (!(HAL_GPIO_ReadPin (C1_PORT, C1_PIN)))   // if the Col 1 is low
    {
        while (!(HAL_GPIO_ReadPin (C1_PORT, C1_PIN)));   // wait till the button is pressed
        return '*';
    }

    if (!(HAL_GPIO_ReadPin (C2_PORT, C2_PIN)))   // if the Col 2 is low
    {
        while (!(HAL_GPIO_ReadPin (C2_PORT, C2_PIN)));   // wait till the button is pressed
        return '0';
    }

    if (!(HAL_GPIO_ReadPin (C3_PORT, C3_PIN)))   // if the Col 3 is low
    {
        while (!(HAL_GPIO_ReadPin (C3_PORT, C3_PIN)));   // wait till the button is pressed
        return '#';
    }

    if (!(HAL_GPIO_ReadPin (C4_PORT, C4_PIN)))   // if the Col 4 is low
    {
        while (!(HAL_GPIO_ReadPin (C4_PORT, C4_PIN)));   // wait till the button is pressed
        return 'D';
    }

}
```

- **Implementation:** In the implementation phase, the 4x4 keypad is wired to the STM32F103C8T6 microcontroller, with connections assigned to GPIO pins (A8 to B4). The ST-Link V2 programmer uploads the firmware, while a USB supplies 3.3V power. The system efficiently scans for key presses, ensuring reliable and responsive operation for password entry or device control.

EXPECTED RESULT

When a key on the 4x4 keypad is pressed, the corresponding character or number is detected by the STM32 microcontroller. This input is processed and transmitted to a connected laptop via a serial interface. The key press is then displayed in real-time on the laptop screen, ensuring accurate and immediate feedback. This functionality can be further extended for applications like password entry, menu navigation, or controlling external devices, offering seamless user interaction and reliable performance.

FINAL RESULT

Upon pressing the keys on the 4x4 keypad, the system occasionally fails to respond as expected, the input cannot processed and transmitted to a connected laptop via a serial interface.

TROUBLESHOOTING: KEYPAD MALFUNCTIONS WITH STM32

After thoroughly examining all potential causes, such as hardware connections, power supply stability, GPIO configurations, and firmware logic, we identified that the issue likely stems from a malfunction in the STM32 microcontroller itself. Despite correct wiring, stable power input, and properly implemented debouncing techniques, the system continues to display irregular behavior, such as delayed responses, incorrect key detections, or complete unresponsiveness.

This malfunction may be due to internal faults in the microcontroller, such as:

- **Damaged GPIO Ports**: Individual pins might be faulty, leading to inconsistent signal readings from the keypad.

- **Corrupted Memory**: Issues in the microcontroller’s flash memory can affect the execution of the scanning routine.
- **Overheating or Electrical Damage**: Excessive current, static discharge, or overheating could have damaged internal components, impairing functionality.
- **Firmware Corruption**: Even after reprogramming via ST-Link V2, persistent issues might indicate deeper problems within the microcontroller’s architecture.

IMPACTS OF THIS PROJECT

This project offers practical applications across various industries, enhancing efficiency and user interaction. It can be integrated into security systems for password entry, improving access control in homes, offices, and secure facilities. In consumer electronics, it can serve as an interface for devices like calculators, home automation panels, or vending machines, providing a simple and cost-effective input method. Additionally, in educational environments, this project helps students understand embedded systems, GPIO interfacing, and coding principles. The system's reliability and adaptability make it a foundational element for developing more advanced control systems, contributing to innovations in automation, security, and interactive technology.

CONCLUSION

The Keypad System using the STM32 microcontroller demonstrates an efficient and practical method for capturing user input in various applications. Despite potential challenges like hardware malfunctions or firmware issues, proper implementation and troubleshooting ensure reliable performance. This project serves as a foundation for developing secure, interactive, and automated systems.