American University of Ras Al Khaimah

Microprocessors Lab

CENG316

Section: 1

Instructor: Engr. Umar Adeel

Lab 3 – Keypad Scanning in C

Abin Raj Devarajan 2022005464

Date of the lab: 09–02–2025

Contents

1 Objective

- Understand the process of scanning a keypad matrix using GPIO.
- Implement an efficient keypad scanning algorithm using software debouncing.
- Display pressed keys dynamically on an LCD, ensuring smooth scrolling.
- Handle multiple keypress scenarios with proper debounce techniques.

2 Equipment Used

- STM32L476G-DISCO board
- 4x4 matrix keypad
- LCD display
- Breadboard and jumper wires
- 2.2k Ohm pull-up resistors
- KEIL tool by ARM

3 Pre-Lab

3.1 Part A – Textbook Readings

Before beginning the lab, students are required to review the following section from the textbook:

- **Chapter 14.9:** Keypad scanning

3.2 Part B – Pre-Lab Assignment

In this lab, we will interface with a keypad using the STM32 microcontroller. The keypad and necessary components such as a breadboard and wires will be provided. Due to the board design, it is recommended to use two breadboards for better connectivity.

We will configure specific pins on **GPIOA** as inputs and **GPIOE** as outputs to facilitate the keypad scanning process.

3.2.1 Configure Port A (Inputs)

We will set GPIOA pins 1, 2, 3, and 5 as digital inputs. These pins are also used by the joystick but will be reassigned for external connections in this experiment.

The configuration is achieved by setting the GPIOA MODER register, where:

- `00` (binary) represents digital input mode.

Register Layout for MODER (Port A)

Register	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
MODER	MODER15		MODER14		MODER13		MODER12		MODER11		MODER10		MODER9		MODER8		MODER7		MODER6		MODER5		MODER4		MODER3		MODER2		MODER1		MODER0	
Mask	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1: Mask in hex: 0x00000000 — Value in hex: 0x00000000

3.2.2 Configure Port E (Outputs)

We will set GPIOE pins 10, 11, 12, and 13 as digital outputs to drive the keypad columns.

The configuration is achieved by modifying the GPIOE MODER register, where:

- `01` (binary) represents digital output mode.

Register Layout for MODER (Port E) Register Layout for MODER (Port A)

Register	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
MODER	MODER15		MODER14		MODER13		MODER12		MODER11		MODER10		MODER9		MODER8		MODER7		MODER6		MODER5		MODER4		MODER3		MODER2		MODER1		MODER0	
Mask	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Value	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2: Mask in hex: 0x00000000 — Value in hex: 0x00000000

4 Lab Procedure

Goal: The end goal of this lab is to be able to push buttons on the keypad, and have the values displayed on the LCD. The display should show the last 6 buttons pressed, and they should scroll off the screen to the left (similar to how a calculator or a telephone would enter the numbers). Software debouncing should be used to make sure no double-presses accidentally happen

4.1 Part A – Connecting the Keypad

1. Connect the keypad to the STM32L4 GPIO pins:
   - GPIOE (Pins 10, 11, 12, 13) for **row scanning**.
   - GPIOA (Pins 1, 2, 3, 5) for **column scanning**.
2. Use external 2.2k ohm pull-up resistors on the column lines.
3. Verify wiring using Figures 1 and 2 from the lab manual.

Figure 1: Keypad

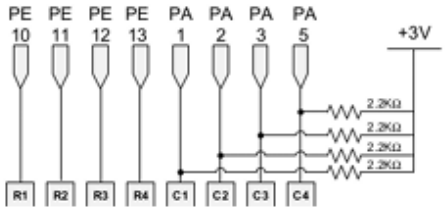

Figure 2: Row and Column code

4.2 Part B – Writing the Keypad Scanning Code

1. Modify `main.c` and add a function `Keypad_Pin_Init()` to configure GPIO.
2. Implement the `keypad_scan()` function to:
   - Write '0b0000' to the rows.
   - Read column values.
   - Print scanned values to the LCD.
3. Implement row-wise scanning and column detection.
4. Use a lookup table to map row-column combinations to keypad values.

4.3 Part C – Displaying Keypresses on the LCD

1. Implement a 6-character keypress buffer.
2. Shift all stored keypresses one position left before adding a new key.
3. Implement software debouncing to avoid repeated key registrations.

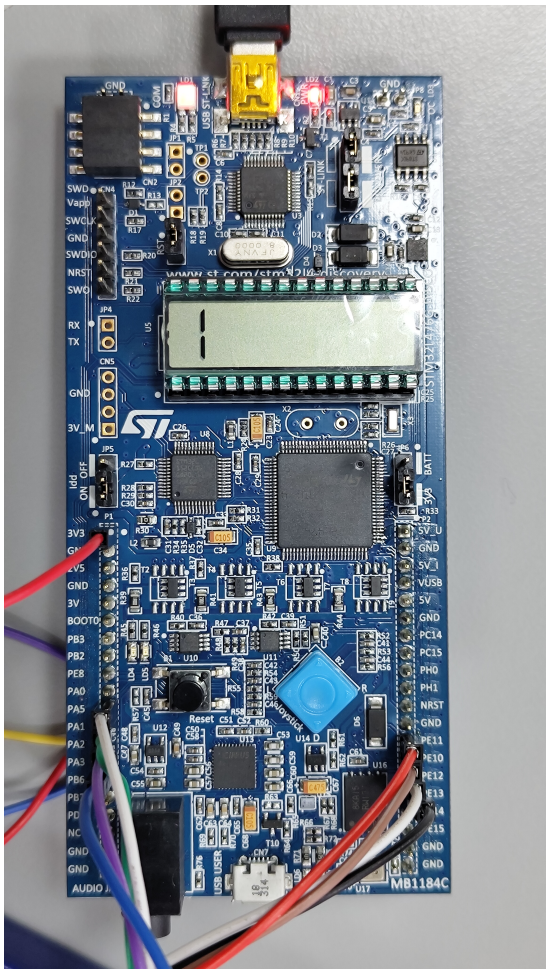

Figure 3: 1

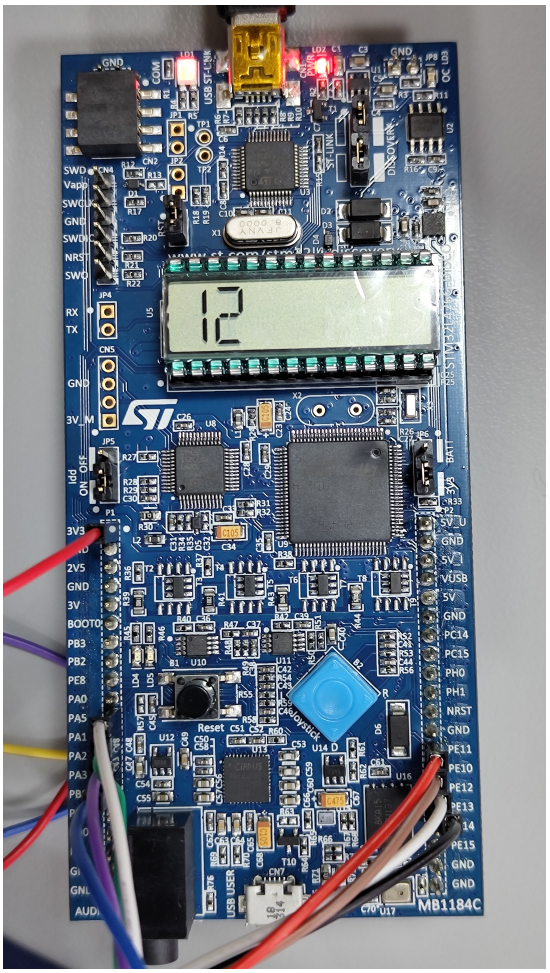

Figure 4: 12

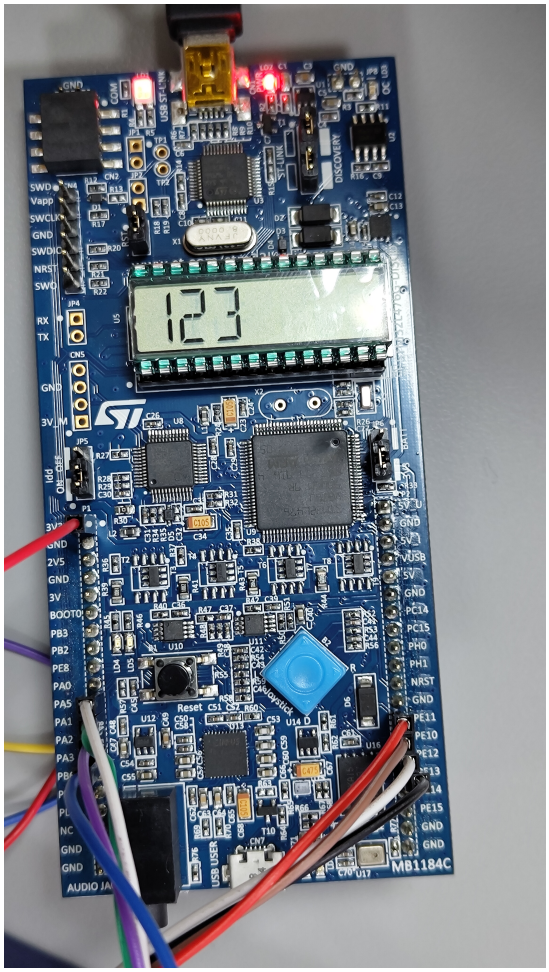

Figure 5: 123

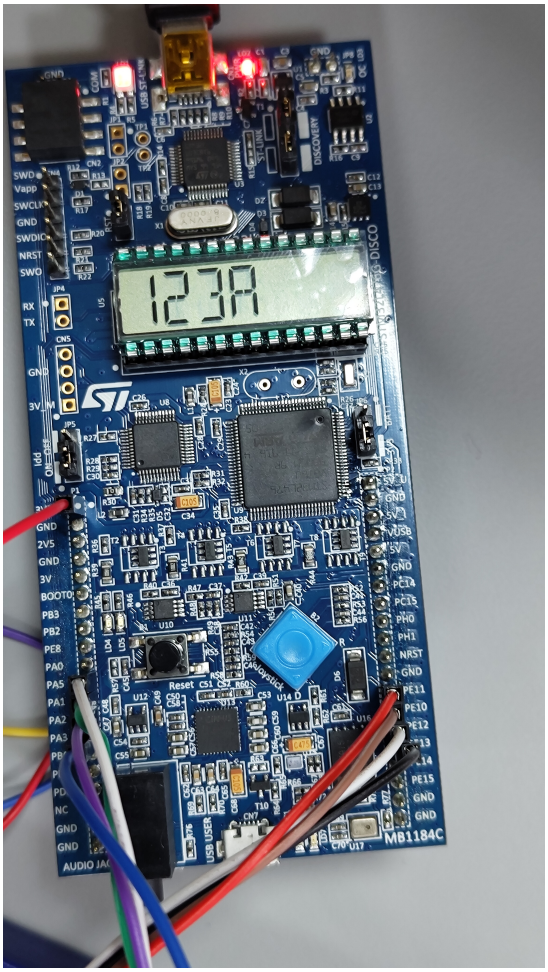

Figure 6: 123A

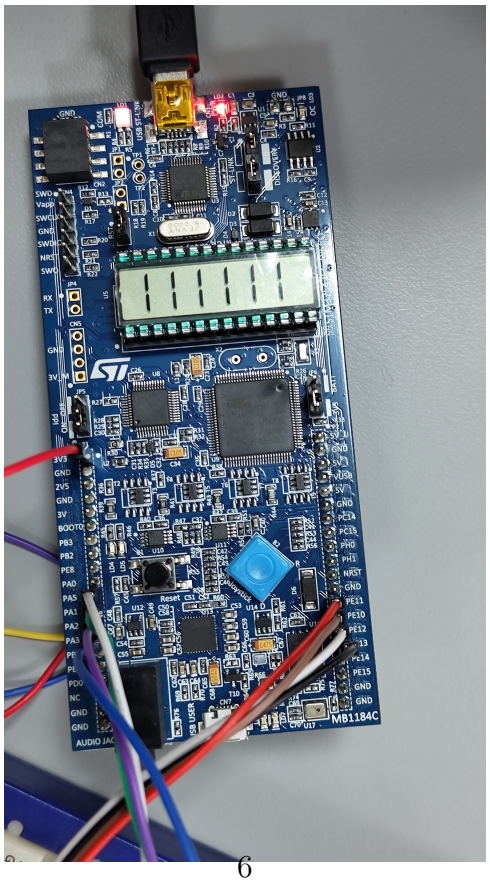

Figure 7: 1111

5 Code Implementation

5.1 Keypad Pin Initialization

Listing 1: Keypad Pin Initialization

```
void Keypad_Pin_Init(void) {
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN | RCC_AHB2ENR_GPIOEEN;

    GPIOE->MODER &= ~(3U<<(2*10) | 3U<<(2*11) | 3U<<(2*12) | 3U<<(2*13)
        );
    GPIOE->MODER |=  (1U<<(2*10) | 1U<<(2*11) | 1U<<(2*12) | 1U<<(2*13)
        );

    GPIOA->MODER &= ~(3U<<(2*1) | 3U<<(2*2) | 3U<<(2*3) | 3U<<(2*5));
}
```

5.2 Keypad Scanning Function

Listing 2: Keypad Scanning

```
char keypad_scan(void) {
    for (int row = 0; row < 4; row++) {
        GPIOE->ODR &= ~(0b1111 << 10);
        GPIOE->ODR |= ~(1 << (10 + row));
        delay(5);

        uint8_t col = (GPIOA->IDR >> 1) & 0b1011;

        if (col != 0xF) {
            for (int c = 0; c < 4; c++) {
                if (!(col & (1 << c))) {
                    return lookup[row][c];
                }
            }
        }
    }
    return 0xFF;
}
```

5.3 LCD Display Update

Listing 3: LCD Display Update

```
void update_lcd(char key) {
    for (int i = 0; i < 5; i++) {
        buffer[i] = buffer[i+1];
    }
    buffer[5] = key;
    LCD_DisplayString(buffer);
}
```

6 C Program for LCD Configuration and Display

The following C program configures and controls the LCD display in an STM32L476 microcontroller. It includes initialization functions, display control mechanisms, and character mapping for the LCD.

Listing 4: LCD Configuration and Display Code

```

#include "lcd.h"
#include "stm32l476xx.h"
#include <stdint.h>

/* ========================
          LCD MAPPING
    ========================
LCD allows displaying information on six 14-segment digits and 4 bars:

  1       2       3       4       5       6
-----   -----   -----   -----   -----   -----
|\|/| o |\|/| o |\|/| o |\|/| o |\|/|   |\|/|   BAR3
-- --   -- --   -- --   -- --   -- --   -- --   BAR2
|/|\| o |/|\| o |/|\| o |/|\| o |/|\|   |/|\|   BAR1
----- * ----- * ----- * ----- * -----   -----   BAR0
*/

// Constant table for capital letters A to Z
const uint16_t CapLetterMap[26] = {
    0xFE00, 0x6714, 0x1d00, 0x4714, 0x9d00, 0x9c00, 0x3f00, 0xfa00, 0
       x0014,
    0x5300, 0x9841, 0x1900, 0x5a48, 0x5a09, 0x5f00, 0xFC00, 0x5F01, 0
       xFC01,
    0xAF00, 0x0414, 0x5b00, 0x18c0, 0x5a81, 0x00c9, 0x0058, 0x05c0
};
```

```
25
26 // Constant table for numbers 0 to 9
27 const uint16_t NumberMap[10] = {
28     0x5F00, 0x4200, 0xF500, 0x6700, 0xEa00, 0xAF00, 0xBF00, 0x04600, 0
       xFF00, 0xEF00
29 };
30
31 // LCD Initialization
32 void LCD_PIN_Init(void) {
33     // Enable GPIO clocks for PA, PB, PC, and PD
34     RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN | RCC_AHB2ENR_GPIOBEN |
       RCC_AHB2ENR_GPIOCEN | RCC_AHB2ENR_GPIODEN;
35
36     // Configure GPIO mode as alternate function
37     GPIOA->MODER &= 0x3FC00FFF;
38     GPIOA->MODER |= 0x802AA000;
39
40     GPIOB->MODER &= 0x00F3F0F0;
41     GPIOB->MODER |= 0xAA080A0A;
42
43     GPIOC->MODER &= 0xFFFC003F;
44     GPIOC->MODER |= 0x0002AA80;
45
46     GPIOD->MODER &= 0x0000FFFF;
47     GPIOD->MODER |= 0xAAAA0000;
48
49     // Configure Alternate Function Registers (AFR)
50     GPIOA->AFR[0] &= 0x00FFFFFF;
51     GPIOA->AFR[0] |= 0xBB000000;
52     GPIOA->AFR[1] &= 0x0FFFF000;
53     GPIOA->AFR[1] |= 0xB0000BBB;
54
55     GPIOB->AFR[0] &= 0xFF00FF00;
56     GPIOB->AFR[0] |= 0x00BB00BB;
57     GPIOB->AFR[1] &= 0x0000FF0F;
58     GPIOB->AFR[1] |= 0xBBBB00B0;
59
60     GPIOC->AFR[0] &= 0x00000FFF;
61     GPIOC->AFR[0] |= 0xBBBBB000;
62     GPIOC->AFR[1] &= 0xFFFFFFF0;
63     GPIOC->AFR[1] |= 0x0000000B;
64
65     GPIOD->AFR[1] |= 0xBBBBBBBB;
66 }
67
68 // Function to display a string on the LCD
69 void LCD_DisplayString(uint8_t* ptr) {
```

```
    for (uint8_t i = 0; i < 6; i++) {
        LCD_WriteChar(ptr++, 0, 0, i);
    }
}

// Function to display a specific name on the LCD
void LCD_Display_Name(void) {
    while ((LCD->SR & LCD_SR_UDR) != 0); // Wait for Update Display
        Request Bit

    // Display "SENERE"
    LCD->RAM[0] |= 0xD782A168;
    LCD->RAM[1] |= 0x00E;
    LCD->RAM[2] |= 0xE5C25278;
    LCD->RAM[3] |= 0x005;
    LCD->RAM[6] |= 0x0000C000;
    LCD->RAM[7] |= 0x008;

    LCD->SR |= LCD_SR_UDR; // Update LCD display
}

// LCD Configuration
void LCD_Configure(void) {
    LCD->CR &= ~LCD_CR_LCDEN;

    // Set bias to 1/3
    LCD->CR &= ~(0x03<<5);
    LCD->CR |= 0x02<<5;

    // Set duty to 1/4
    LCD->CR &= ~(0x07<<2);
    LCD->CR |= 0x03<<2;

    // Set contrast and pulse on period
    LCD->FCR |= LCD_FCR_CC | LCD_FCR_PON;

    // Select internal voltage source
    LCD->CR &= ~LCD_CR_VSEL;

    // Wait for LCD to be ready
    while ((LCD->SR & LCD_SR_FCRSR) == 0);
    LCD->CR |= LCD_CR_LCDEN;

    while ((LCD->SR & LCD_SR_ENS) == 0);
    while ((LCD->SR & LCD_SR_RDY) == 0);
}

```

```c
// LCD Initialization Wrapper
void LCD_Initialization(void) {
    LCD_PIN_Init();
    LCD_Clock_Init();
    LCD_Configure();
    LCD_Clear();
}

// LCD Clock Initialization
void LCD_Clock_Init(void) {
    RCC->APB1ENR1 |= RCC_APB1ENR1_PWREN;
    if ((PWR->CR1 & PWR_CR1_DBP) == 0) {
        PWR->CR1 |= PWR_CR1_DBP;
        while((PWR->CR1 & PWR_CR1_DBP) == 0);
    }

    RCC->BDCR &= ~(RCC_BDCR_LSEON | RCC_BDCR_LSEBYP);
    RCC->BDCR |= RCC_BDCR_BDRST;
    RCC->BDCR &= ~RCC_BDCR_BDRST;

    while ((RCC->BDCR & RCC_BDCR_LSERDY) == 0) {
        RCC->BDCR |= RCC_BDCR_LSEON;
    }

    RCC->BDCR &= ~RCC_BDCR_RTCSEL;
    RCC->BDCR |= RCC_BDCR_RTCSEL_0;
    RCC->APB1ENR1 &= ~RCC_APB1ENR1_PWREN;
    RCC->APB1ENR1 |= RCC_APB1ENR1_LCDEN;
}

// Function to clear LCD display
void LCD_Clear(void) {
    while ((LCD->SR & LCD_SR_UDR) != 0);
    for (uint8_t counter = 0; counter <= 15; counter++) {
        LCD->RAM[counter] = 0;
    }
    LCD->SR |= LCD_SR_UDR;
}
```

7 Post-Lab Questions

1. **Can your code correctly handle multiple keys pressed at once?**

    - **Answer:** No, the current approach detects only one key at a time. Handling multiple keypresses would require additional logic to track simultaneous key

states.

2. **How do you debounce keypresses?**
   - **Answer:** Debouncing is implemented by:
     - Adding a small delay after detecting a keypress.
     - Waiting for the key to be released before registering another press.
3. **Why are external pull-up resistors used instead of internal ones?**
   - **Answer:** Internal pull-ups are around 40kohm, which is too weak for reliable operation. The 2.2kohm external pull-ups ensure proper voltage levels.

8 Conclusion

This lab successfully demonstrated the implementation of a keypad scanning system using the STM32L476G-DISCO microcontroller, integrating hardware and software techniques to detect and display keypresses dynamically on an LCD screen. The experiment provided hands-on experience in configuring GPIO pins for input and output, utilizing pull-up resistors, and implementing a software debouncing mechanism to ensure reliable key detection.

Key takeaways from this lab include:

- Understanding the principles of matrix keypad interfacing and GPIO configuration for effective row-column scanning.
- Developing a real-time keypad scanning algorithm that efficiently detects keypresses and prevents ghosting effects.
- Implementing software debouncing techniques to eliminate false triggers and ensure accurate input recognition.
- Designing a dynamic LCD update mechanism that allows for smooth scrolling of keypresses, emulating behavior found in calculators and electronic keypads.

The results of this lab demonstrate the effectiveness of embedded system programming in handling human-machine interfaces (HMI) and real-time interactions. The successful integration of the keypad and LCD display paves the way for more complex applications, such as password-protected access systems, interactive menu navigation, and embedded UI development.

Future improvements could include multi-keypress detection, interrupt-based scanning, and event-driven input processing, enhancing the system's efficiency and responsiveness. This lab has provided a strong foundation for implementing real-world embedded applications that rely on efficient input handling and user interaction.