**ECE0202: Embedded Systems and Interfacing**

**Lab 4: Keypad Scanning (in assembly)**

**Due: 3/21/21 at 11:59pm**

**Objectives**

* **Understand I/O multiplexing technique**
* **Be familiar with keypad scanning algorithms**
* **Implement software debouncing**

**Pre-Lab Reading**

* **Chapter 14.9 Keypad Scanning**

**Deliverables – total 100 points**

* **(40 points) – demonstration of keypad scanning that displays the pressed character on the Tera Term.**
* **(35 points) – Code submission. Code should use software debouncing and be well-written and commented.**
* **(25 points) Submission of the pre-lab register tables and post-lab questions. Indicate how each group member participates and contributes to the lab at the end of the lab report.**
* **(10 points extra credit) Display the last pressed 6 characters, in order, on the Tera Term.**
* **Indicate each group member’s Participation and Contribution.**

**Please submit your code as \*.s files and your schematic as a pdf**

**Keyboard Interface**

The 4x4 keypad used in this lab requires 8 pins (4 row pins and 4 column pins). In this lab, the connection between the keypad and the discovery kit is shown in the following table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Row | R1 → PC0 | R2 →PC1 | R3 →PC2 | R4 →PC3 |
| Column | C1 → PB1 | C2 → PB2 | C3 → PB3 | C4 → PB5 |

All pins of the input port (C1, C2, C3, and C4) are pulled up to 3V via 2.2k resistors already placed on the Discovery board; however, the output port pins (R1, R2, R3, and R4) will require us to configure pull-up resistors. Within the processor, each GPIO pin can be pulled up via an internal resistor (between 20 and 55k), but the internal pull-up current capability is too weak, and therefore an external pull-up resistor is required, as drawn in Fig. 1.

When looking at the front side of the keypad, the pins on the connector from left to right are:

R1 – R2 – R3 – R4 – C1 – C2 – C3 – C4

Diagram

Description automatically generated

Figure 1- Picture and schematic of the keypad

The maximum current a GPIO pin can source or sink is 20 mA. When calculating the value of external pull-up resistors, make sure that the current does not exceed 20 mA. For example:

On the Nucleo board, all pins in the input port (PB1, PB2, PB3, and PB5) are connected to ground via a 100 nF capacitor. A very short delay should be added before reading the input port, as seen later in the software flowchart.

**ASCII**

In order to write to the Tera Term, you must store the code for an ASCII character into a memory location. In lab 1, the string to be displayed on the Tera Term is stored at a memory location called “str”. This can also be used as the memory location that stores the character you display in this lab.

To display a character on the Tera Term, you must store the associated ASCII code in the memory location “str”, load the memory address of “str” into r0, and then run the instruction *BL USART2\_Write*.

The following table gives ASCII codes for many characters. Note that these are in decimal!

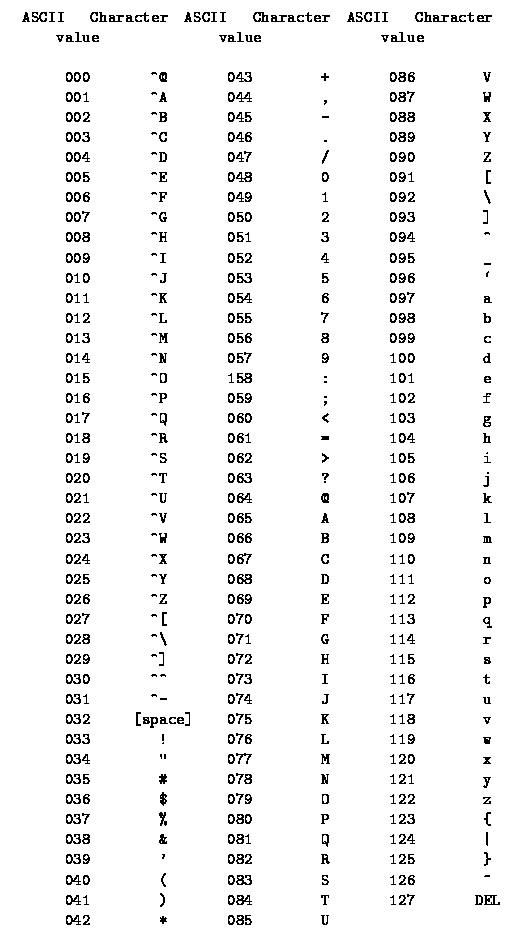


Figure 3 - ASCII character codes. From https://ee.hawaii.edu/~tep/EE160/Book/chap4/subsection2.1.1.1.html

**Software Flowchart**

The following software flowchart is a modified version of that shown in textbook chapter 14.9, and should be used as a general guide for writing the program used in this lab.

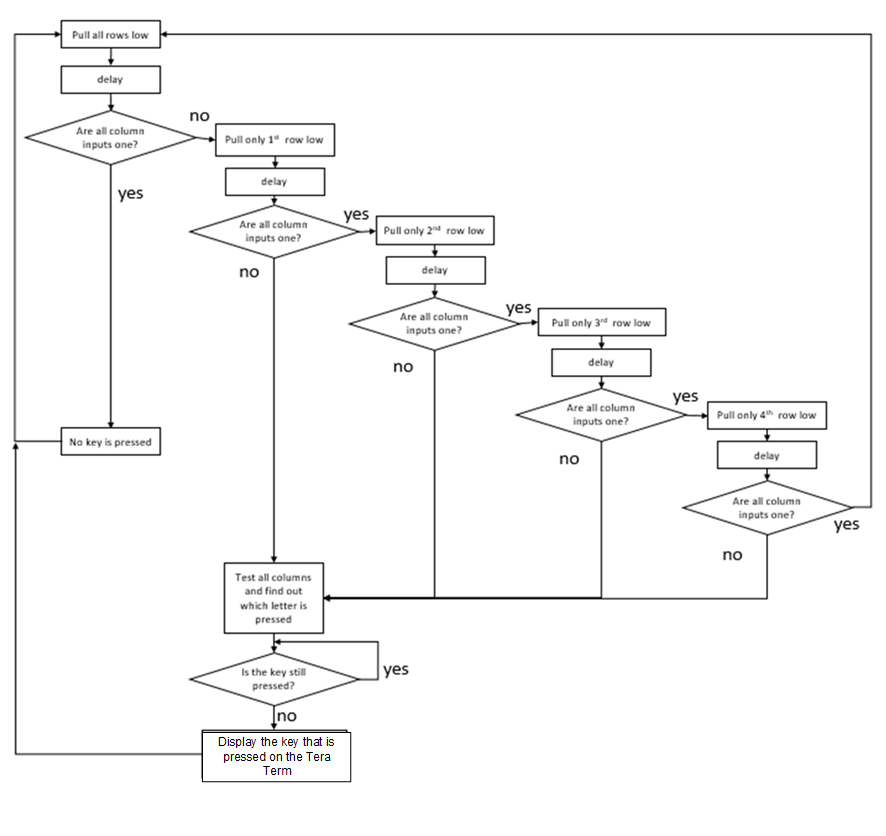
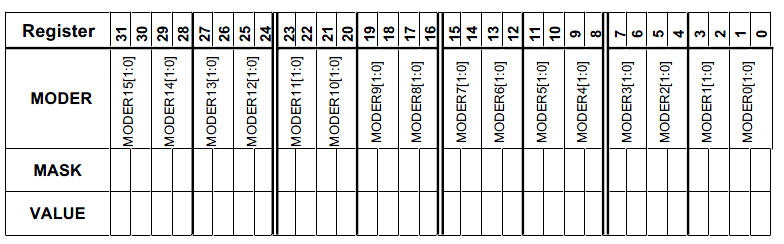


Figure 4 - Software Flowchart for the keypad scanning algorithm

**Pre-Lab Register Tables (5 points)**

**Configure Port C: Pin 0, 1, 2, and 3 as Digital Output**

GPIO Mode: Digital Input (00), Digital Output (01), Alternative Function (10), Analog (11)

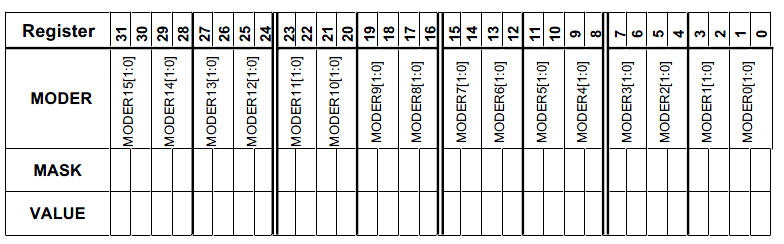


GPIOC Mode Register MASK Value = 0xFF (in HEX)

GPIOC Mode Register Value = 0x55 (in HEX)

**Configure Port B: Pin 1, 2, 3, and 5 as Digital Input**

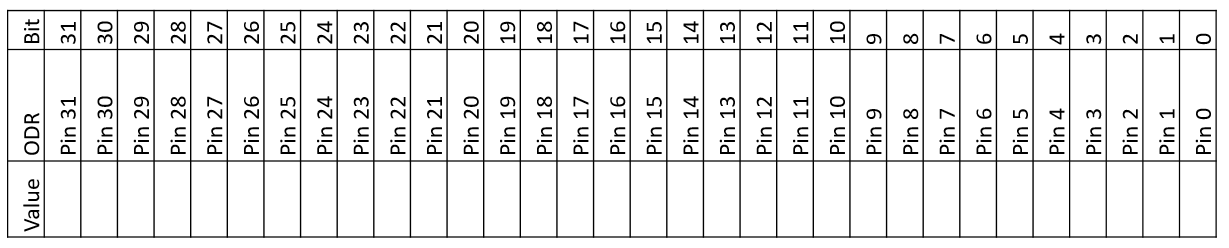
GPIO Mode: Digital Input (00), Digital Output (01), Alternative Function (10), Analog (11)



GPIOB Mode Register MASK Value = 0xCFC (in HEX)

GPIOB Mode Register Value = 0x303 (in HEX)

**Write to Port C: Pins 0, 1, 2, and 3 connect to the rows of the keypad**



**Value written to PORTC ODR in order to pull down all rows: 0x0 (in HEX)**

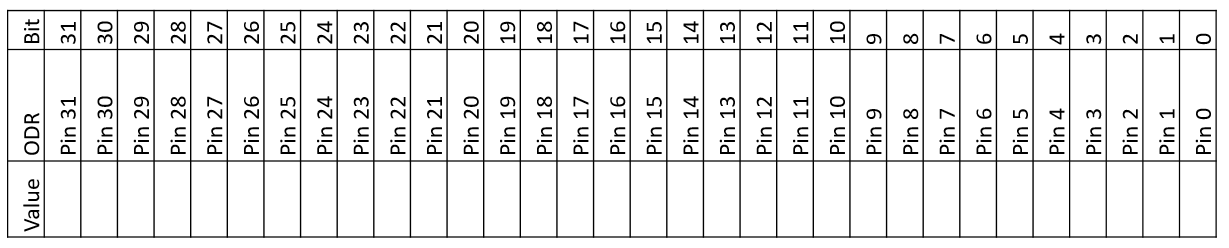
**Value written to PORTC ODR in order to pull down row 1: 0xE (in HEX)**

**Value written to PORTC ODR in order to pull down row 2: 0xD (in HEX)**

**Value written to PORTC ODR in order to pull down row 3: 0xB (in HEX)**

**Value written to PORTC ODR in order to pull down row 4: 0x7 (in HEX)**

**Read from Port B: Pins 1, 2, 3, and 5 connect to the columns of the keypad**



**Mask to check if a button from column 1 has been pressed: 0x02 (in HEX)**

**Mask to check if a button from column 2 has been pressed: 0x04 (in HEX)**

**Mask to check if a button from column 3 has been pressed: 0x08 (in HEX)**

**Mask to check if a button from column 4 has been pressed: 0x20 (in HEX)**

**Post-Lab Questions (20 points)**

Please include answers to the following questions with your submission of the pre-lab register contents:

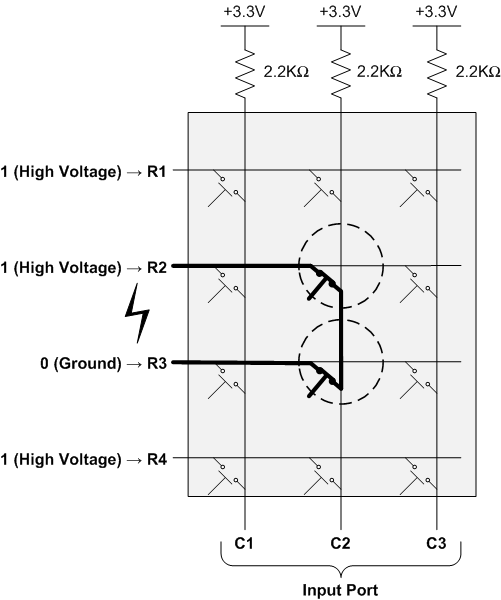
1. How is software debouncing implemented in your program? (3 points)

We implemented software debouncing by placing delays after each time the row was changed and by putting a loop within the display key function that continued to loop until the none of the buttons were being pressed. This was determined by checking the IDR and comparing it to the expected value (when no buttons were pressed).

1. What do we mean when we say that the STM32L4’s internal pull-up resistors are too weak for this application? (5 points)

The resistors that are within the system have higher resistances than 2.2k ohms. This would result in a voltage that is lower than we would need to achieve the desired threshold voltage through the ciruit.

1. When multiple keys are pressed, there could be a short circuit (as shown in the figure). How to configure the output GPIO to avoid this scenario? (7 points)



The output GPIO pins can be configured using an open drain so that they default to high impedance and therefore do not allow short circuiting.

1. In the following 2 cases, can the scan algorithm correctly detect all keys pressed? If so, how to modify the flowchart (figure 4) of the scan algorithm. If not, explain the reason. (5 points)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Case1 Case2

Means the keys that are simultaneously pressed.

Currently, the scan algorithm cannot detect all the keys pressed at the same time. If we wanted to be able to do this, we would need to modify the current flowchart that we have been following. If we find a row and search through its columns to find a pressed character, after displaying that character, we would want to jump back into checking the remaining rows for pushed buttons instead of delaying and looping back to the top. That would get case 1 to work. In order to get case 2 to work, we would need to make sure that we are not stopping after we find a character in a column and instead that we iterate through each and every column for each and every row.

**Participation and Contribution**

Please indicate the participation and contribution for each group member using the following table.

|  |  |
| --- | --- |
| Name | Participation and Contribution |
| Mark | Building, coding, and lab |
| Hayden | Building, coding, and lab |