ECE271: Microcomputer Architecture and Applications — University of Maine

Lab #3: Keypad Scanning in C

Week of 11 February 2019

Goals

1. Be familiar with keypad scanning algorithms.
2. Understand software debouncing.

Pre-lab

1. Complete the pre-lab before attending lab. The pre-lab is in a separate pdf file, found on the website.
2. Be sure to bring a breadboard and jumper wires to the lab. The keypad will be provided.

Lab Procedure

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.

Part A – Connect the Keypad

1. For this lab we will be attaching to the GPIO pins via the external pins on the STM32L4 board.
2. An easy way of doing this is pushing the pins into a breadboard, and then putting components on the breadboard. It might help to have two breadboards, one for each side of the board.
3. You will need some 2.2k resistors (red red red) to act as external pullups. (The internal pullups are typically around 40k and not strong enough for this application).
4. Connect the pins from the keypad to the keypad matrix as shown in Figures 1 and 2.
5. We will use GPIOE pins 10, 11, 12 and 13 for row scanning and GPIOA pins 1, 2, 3 and 5 for column scanning. You might recall from Lab #1 that these GPIOA pins are also connected to the joystick, and thus means there are some capacitors connected to the lines. This should not affect the lab.

   We are re-using these lines because there are a limited set of GPIO pins we can use and many of them are in use by the LCD.

Part B – Getting the Keypad Set Up and Working

1. Use your code from Lab#2 as a base. Copy it over to a Lab#3 directory. Leave `LCD.c` as-is, but modify `main.c`

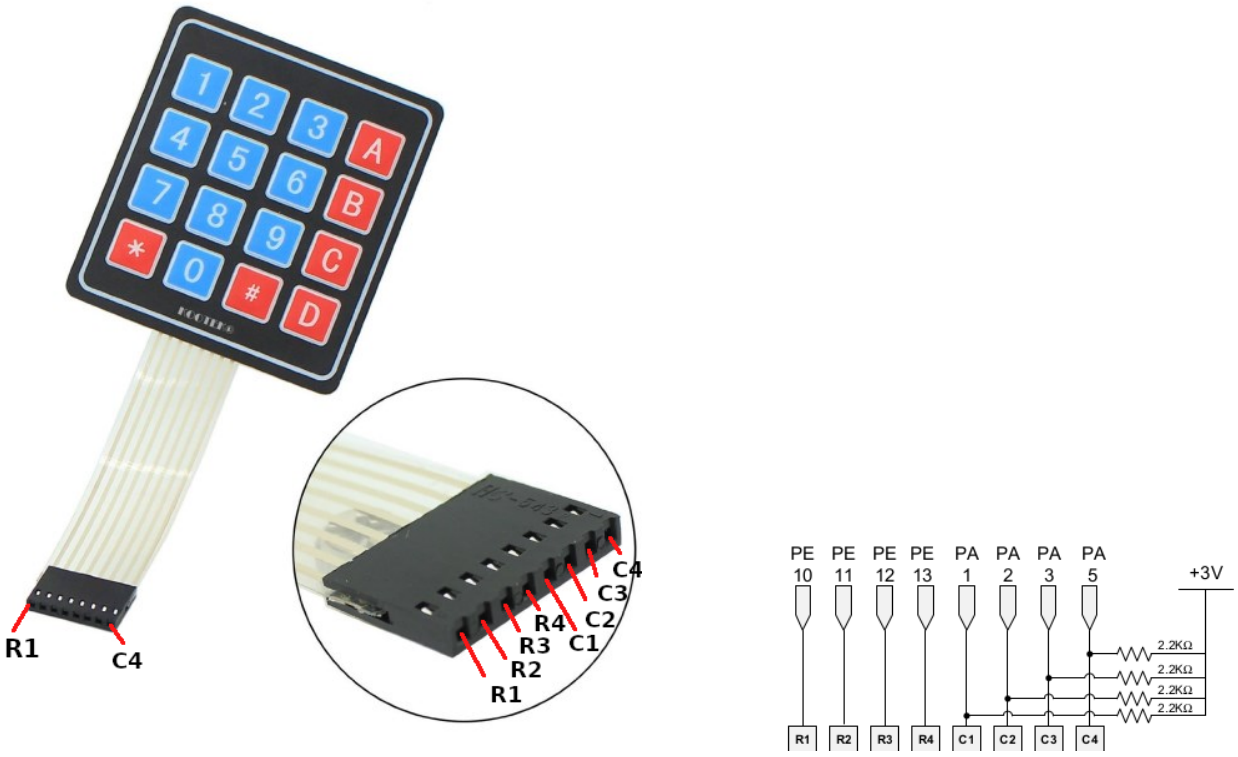

Figure 1: Keypad pinout. If your keypad is only 4x3, the connections are the same, just with C4 missing. Be sure to pull PA5 up to 3.3V even if your display does not have a C4 line.

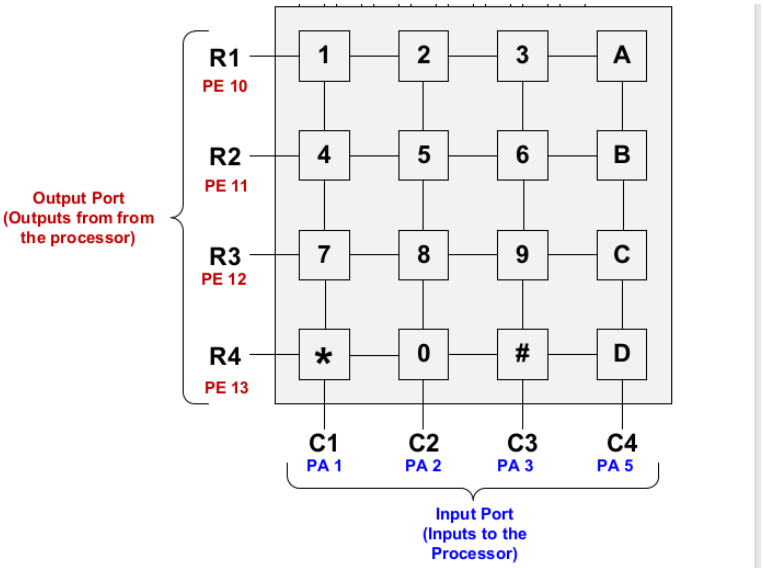

Figure 2: Keypad Matrix as used in Lab3. If you only have a 4x3 display this will look the same, but C4 (the ABCD column) will not be there.

2. Add a `Keypad_Pin_Init()` function.

   (a) This should first enable the GPIOA and GPIOE clocks.

   (b) Next set GIOA pins 1, 2, 3, and 5 as digital inputs and GPIOE pins 10, 11, 12, and 13 as digital outputs. (These are the values you calculated in the pre-lab).

3. Now set up the scanning code. Create a `keypad_scan()` function in `main.c` that handles the scanning, and add an infinite loop in `main()` that calls `keypad_scan()` and updates the display using your `LCD_Display_String()` from Lab#2.

   (a) The textbook has some template code for this in Chapter 14.9 that you can follow.

4. It might be best to break this task up into small chunks to verify things are working.

   (a) First have `keypad_scan()` write 0b0000 to the rows. Then read the column input. Print the raw 4-bits of result you read from the inputs to the LCD. Remember, you can convert a 0/1 value to ASCII for displaying by adding '0' to it. If you print the value you read to the display, you should be able to see the result change as you press the buttons.

   HINT: remember that the GPIO pins are not consecutive, so you need to grab PA5, PA3, PA2, and PA1 (bits 5, 3 2 and 1) and perhaps shift and mask them separately to interpret the results.

   If no key is pressed, return 0xff from the function.

   (b) Next, once you've verified your wiring is correct and that the buttons work, it is time to do a column scan and figure out in what column the keypress is happening.

   You can do this by taking the value from the previous step, and checking which bit is 0. If PA5 is 0, then it is Column4, if PA3 is 0, then it is Column3, etc. You can print this to the display if you want to verify you are getting the column you expect.

   (c) After that, it is time to walk through the rows to find which row is being held down. Walk through each of the 4 rows, setting just the one bit to zero in each one. `0b0111`, `0b1011`, `0b1101`, `0b1110`. Note, again, that these are not consecutive bits.

   Also, you will probably need to delay slightly after setting the row before you read out the column value otherwise you might get old results.

   Walk through the 4 row values (also remembering that you should probably clear all the bits first, before ORing in the pattern you want). Read out the column results, and if the column result is not all 1s it means you've found the row that has a key being held down. Again you can print the row to the LCD to help debug.

   (d) Finally, once you can determine the column and row, then return a value from the keypad. One way to do this is have a lookup table like in the book. If it is row=0, col=0, return '0' (ASCII 0). If it is row=0,col=1, return '1'. If it is row=1,col=1, return '5', and so forth.

   As always, you can display this to the LCD to verify it is working.

Part C – Displaying button presses on the LCD

1. Now that we can detect a keypress, we want to keep track of the keypresses and display them to the LCD.

2. We want a keypress buffer than stores the last N keypresses. This only needs to be 6, but for some of the "something cool"s you might want to make it bigger. You will want to initialize this to being empty (' ' characters) at startup.

3. Next, when a keypress happens, you want to move all of the current keypress buffer to the left by one. An easy way to do this is a loop.

4. Then put your current keypress in the right-most position.

5. Finally display the buffer to the LCD.

6. One thing you might find is that you press a key it will quickly fill the buffer with the key you are pressing. You will need to add some code that waits until the key is let go before sending only a single keypress to the buffer. One way of doing this is remembering the last key pressed, and only process a keypress if this changes (either to another key, or to the 0xff no-keypress value).

   Also, it is possible that the keypresses are noisy and send double results, so additional debouncing might be necessary.

Part D – Something Cool

Do something cool! You can come up with something on your own, but here is a list of ideas you can use.

1. When a key is pressed for a long time, repeat the value being printed. (This is sometimes called auto-repeat)

2. Use the '*' key to delete the previous input.

3. Use the '#' key to repeat previous input.

4. Detect if multiple keys are pressed simultaneously.

5. Come up with some way to enter letters using the keypad. For example, use the old-fashioned touch-tone phone method of pressing '2' for A, but pressing '2' twice quickly for 'B', three times for 'C', etc.

Lab Demo

1. Submit your code
    - Complete a README with the post-lab answers.
    - Make sure the code is properly commented.
    - Check your code and README into your gitlab tree.
2. Demo your implementation to your lab TA.
    (a) Do all keypresses show up on the display?
    (b) How do you debounce the keypresses?
    (c) Be able to explain your keyscan algorithm.
    (d) Why did we need to use external pullup resistors instead of the built in ones?

Post-Lab

- Place your answers to the question in a file Readme.md
- Submit with your code via the gitlab server.
- Questions:
    1. Can your code correctly handle if multiple keys are pressed at once? Why or why not?