TA Andrew Gerber 10/17/2023

Lab 05 Report

Benjamin Meads
A02323437
Nick Templeton
A02268123
Richard Snider
A02298443

Objectives

This lab's objective was to introduce the keyboard and integrate it with the LCD configuration we used in Lab 4. Our objective was to be able to press a keypad digit and have it display on the LCD one character at a time. We needed to have it printed on the first line until it was filled up. Before switching to the second line. We needed to have a way to initialize both the LCD and the Keypad in their own files. Then be able to use headers to bring everything together. Another point we needed to complete was to wire up the keypad according to the diagram shown below (Fig 1.) This used the resistors provided in the lab kit. Then one end goes to 3.3V power, another end goes to the keypad, and another end goes to the respective GPIO pins.

Procedure

Our procedure for this lab started off as us configuring what we knew needed to be initialized first. The LCD initializations were left as is. However, we decided to configure the GPIO pins we were going to use specifically for the keypad. GPIO B pins 6 through 9 were configured to be inputs. While GPIO B pins 10 through 13 were configured as outputs. We also set our pins to be open-drain as to protect our board from any short circuit the keypad may have done. Then we set up the actual circuit. The circuit needed to have each input pin of the keypad to be bridged. With one wire going across a 2.2 KOhm resistor and then a 3.3 power source gotten from the board. Then the other bridge of it heading to the designated GPIO port. Once we had the wiring correct we moved on to the logic of our program. We spent a long time deciding whether or not to follow the instructions presented during class in the slides. But in the end, we decided to configure our own logic. So, instead of reading an input from the columns then cycling a low voltage through each of the rows. We decided to constantly cycle through the rows with a low voltage one at a time. Then at any point if we were to read a low voltage on any of the inputs. We were able to determine which of the rows

was currently set to low. As well as the input that we were reading low. This was able to get us our row and column associated with the button. Then we were able to use this information to plug into an array of arrays variable we had that was set to the correct character that button represented. Another reason we decided to do it a different way, is to add another level of protection to our board. By only having one row set to low at a time. We could make sure that any two buttons pressed at the same time wouldn't receive a short circuit. After our lab TA told us our logic behind this was sound. We went head first into solving and figuring out what we needed to do. This entire process took up most of our lab time, going through the logic, and the code to determine where we may have mixed up our process and we had to debug through many errors.

Eventually, we were able to get a character to show up on the LCD screen. However, the character was wrong. It seemed the character displayed was inverse both horizontally and vertically to the character we pressed on the keypad. This was a simple fix as we just edited our case statements to read an opposite value and we were able to get the correct character on the screen. Then our next issue was that our LCD was only displaying one character at a time. Therefore we had to edit our main.c file to represent a list of characters one after the other. We attempted to pass off at this point with the lab TA. However, we didn't configure the line to break at the correct point. Causing the characters to simply overflow the first line and the LCD didn't go to the next line. After a simple fix to an if statement made sure the characters were displayed on the second line once the first line was filled up. At this point, we were able to pass off the lab.

Results

At the end of the lab, we were able to get the keypad to work correctly. Displaying the correct characters pressed on the keypad onto the LCD screen. Once the first line of the LCD was filled up, It was then displayed on the second line. Even though it took our team a bit longer than we were expecting, we were able to get the lab completed to all of the specifications listed in the requirements.

Figures

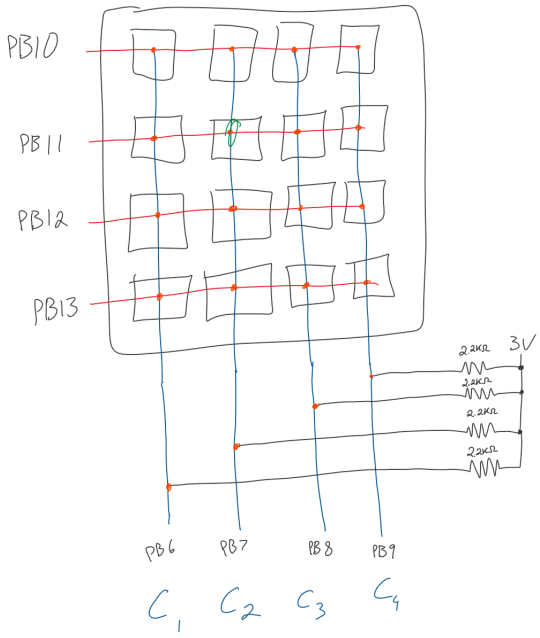

Fig 1: Keypad Schematic

Conclusion

This lab took our team a bit of time to figure out. The majority of our complications were spawned by trying to figure out our own way of implementing the logic of the code. We also believe that the way we did it was more safe, and arguably a more efficient way of determining the correct key press. It made for perhaps a more complicated and difficult way of completing the lab. But now that we have a finished product. We're happy and satisfied with the way that we completed it. We now have experience with the keypad integrating with our microcontroller.

Program

Keypad.c

```
/*
PseudoCode:

create a character array of arrays to hold the values of our keypad at
the correct positions
Keypad initialization
     Set GPIOB pins 6-9 to be input
```

```
     Set GPIOB pins 10-13 to be output
     Set GPIOB pins to be open-drain

ReadAllRows
     Scan through all of the rows and set the value at that row to be
low. This row will be our i-value
     Call the Read row function to get our j value
     We continuously cycle through all of the rows, setting each one to
be low, then reading the columns checking if any of the columns are low.
     If we read a low value through the columns, we know which row is
set to low, and which column is reading low. This gives us an i & j
value to put into our array of arrays
     If we get a valid value, return that row

Scan Row
     Using the case statement, check and set low the passed in a row.
     Then using another case statement, check the columns for input
*/

#include "stm321476xx.h"
#include "keypad.h"
#include "LCD.h"

static unsigned char keypad_chars[4][4] = {
      { '1', '2', '3', 'A' },
      { '4', '5', '6', 'B' },
      { '7', '8', '9', 'C' },
      { '*', '0', '#', 'D' }
};

void keypad_init() {
      //Sets PB0-5 to be outputs
      //writing to dummy variable
      uint32_t new_mode = GPIOB->MODER;
      //changing dummy variable
      new_mode &= (0XF0000FFF);
      new_mode |= (0X05500000);

      GPIOB->MODER = new_mode;

      uint32_t new_otyper = GPIOB->OTYPER;
      new_otyper |= 0x00003C00;
      GPIOB->OTYPER = new_otyper;
}

unsigned char readAllRows()
{
      for(int i = 0; 1; i = (i + 1) % 4) {
```

```
            int j = readRow(i);
            if (j != -1) {
                  return keypad_chars[i][j];
            }
      }
}

int readRow(int i)
{
      unsigned int row = 0;
      unsigned int column = 0;
      int j = 0;

      if (i==0){
            row = 0xe;
      }
      else if (i==1){
            row = 0xd;
      }
      else if (i==2){
            row = 0xb;
      }
      else if (i==3){
            row = 0x7;
      }
      else{
            row = 0xf;
      }

      uint32_t temp_GPIOB = GPIOB->ODR;

      //Clear pins 10-13 to 0 so we can change our ODR output bits
      temp_GPIOB &= (0xFFFFC3FF);
      //Write row to pins 10-13
      temp_GPIOB |= (row << 10); //shifting row to be in the proper
position (10-13)
      //Write dummy variable back to GPIOB ODR
      GPIOB->ODR = temp_GPIOB;

      delay_ms(4);

      temp_GPIOB = GPIOB->IDR;
      temp_GPIOB &= (0x000003C0);
      column = temp_GPIOB >> 6;

      switch(column)
      {
```

```
                case 0xe: j = 0;

                          break;

                case 0xd:   j = 1 ;

                          break;

                case 0xb: j = 2;

                          break;

                case 0x7:   j = 3;

                          break;

                default: j = -1;
          }

          return j;
     }
```

Keypad.h

```
     #ifndef __STM32L476R_NUCLEO_KEYPAD_H
     #define __STM32L476R_NUCLEO_KEYPAD_H

     void keypad_init(void);
     unsigned char readAllRows(void);
     int readRow(int i);

     #endif
```

Main.c

```
     #include "stm32l476xx.h"
     #include "LCD.h"
     #include "keypad.h"

     int main(void){

          LCD_Init();
          LCD_Clear();
          keypad_init();

          LCD_WriteCom(0x80);

          uint32_t count = 0;
          while (1) {
                unsigned char out = readAllRows();
```

```
                count++;
                if (count == 17) {
                      LCD_WriteCom(0xC0);
                } else if (count >= 33) {
                      continue;
                }

                LCD_WriteData(out);

                delay_ms(100);
          }

          /*
          step1: Set output PB10-13 (rows) to 1111
          step2: Cycle through the rows. Always be looking and reading
     input from PB6-9 (input data)
                                -When any of PB6-9 (input data) go low,
     some key in that row has been pressed. Go to step 3.
                                -If all are high, keep doing this step.
          step3: Check every column in the row.
                                -

          for(i=0;i<=4;i++)
          {
          scanrow(i);

          }
                                          -If PB6-9[i] (col) goes low again,
This tells us that our button is row[i],col[j] is pressed

     We will need to set the value of each row and col pair to the
character value.
     */
}
```