TA: Matt Blake
10/18/2018

Lab 5 Report

Andrew Higbee A01648554
Brandon Reed A01577421
Benjamin R. Smith A01521267

Objectives:

The purposes of this lab are to gain experience in:
1. Interfacing the microcontroller with a KeyPad. Liquid Crystal Display (LCD)
2. Build upon previous work using the Liquid Crystal Display (LCD).

Procedure:

After reading through the lab instructions we developed a design for our keypad system. Our initial design (see Figure 1) had R1-R4 set as open-drain outputs with pull-up resistors on pins PE10-PE13 and C1-C4 as inputs on PA0-PA3. We discovered that we would not need the pull-up setting on the outputs and changed our design accordingly.

We composed code to yield these GPIO values. We set the GPIOE mode register for pins 10-13 as outputs (01), the output type register for these pins as open-drain (1). We set the GPIO mode register for pins 0-3 as inputs (00). Refer to appendix A for code.

We simultaneously built the physical prototype of our system using the discovery board, a solderless breadboard, the keypad, and four 2.2KOhm resistors. The resistors were connected on the breadboard between the positive voltage supply and the inputs. Ribbon cables were used to connect the breadboard and the microcontroller; lines for voltage from the power supply were connected as well as ground from the microcontroller. Each of the outputs was connected to ground.

After this we developed our algorithm for determining the pressed key. We initially set the GPIOE ODR to 0 and then scanned the input on GPIOA. We copied the entirety of GPIOA IDR to a variable and then ANDed (&= in c) the value with 0x0000000F in order to only view bits 0-3. We then used case statements to handle 5 scenarios: one of inputs 0-3 being pressed, and the fifth of nothing being pressed.

The case statement for each column called a function (findLocation) for then determining the row. This function changed the values of the GPIOE ODR one row at a time to zero (with the other rows at 1), rescanning the GPIOA IDR each time. If the incorrect row was set to 0 then the IDR would column value would change to 1 (because the output was no longer sinking current). If the new value read from the IDR matched the initial IDR, the correct row would be found.

Our column finding case statements were functioning correctly, but our row finding function was not. The correct column of the keypad was being retrieved, but when the ODR values were changed one at a time, the IDR was not changing at all. We made sure appropriate delays were put in place after changing the ODR value.

After reviewing our design from the ground up and troubleshooting with a lab TA we determined that we were incorrectly connecting our output pins to ground. We were effectively bypassing the capacitors on the discovery board. After removing these connections, we began seeing the IDR values change the way we had originally expected them to.

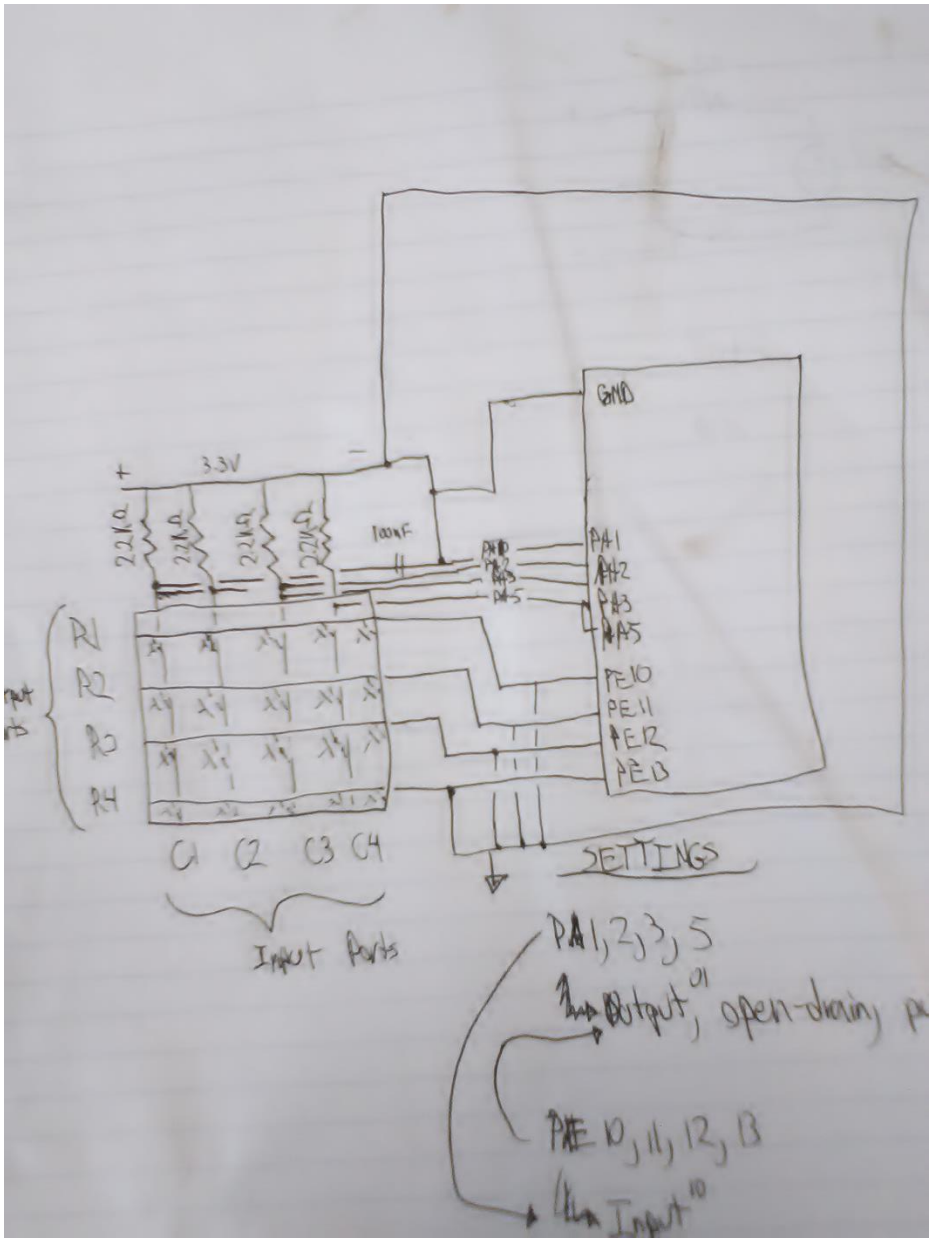
After determining the correct keypress, we used a function (gridToChar) to convert the column and row value to a character. The character was then passed to the displayMessage function to display on the LCD.

Results:

We successfully implemented a keypad on our microcontroller discovery board. We successfully displayed six characters on the LCD of the board. We increased our understanding of the use of an open-drain and pull-up resistors.

Figures:

Figure 1: Initial design.



Conclusion:

This lab taught us how to use a keypad with a microcontroller. We also were able to build on the previous LCD lab and use the keypad to display a character on the LCD. We encountered significant problems based on an incorrect ground, but were able to resolve all issues and produce a functioning system.

Appendix A, LCD.c

```
#include "LCD.h"
#include "Keypad.h"
#include "stm32l476xx.h"
#include <stdint.h>
#include <stdbool.h>

int location[2];//columns then rows of the keypad
int ColumnResultIDR; //record column from IDR
int prevColumnResultIDR;
uint8_t message[6]; //Up to 6 characters to display on LCD
int msgCount = 0; //Used in displayMessage
bool findFlag = 0; //True when a keypad value has been determined

void initKeypad()
{
    /*
    enable clock GPIO E
    for the output pins PE[10:13]
    set MODER to output mode                        01
    set otyper to output open drain    1
    set pupdr to no pull up/pull down 00
    for the input pins PA[0:3]
    set MODER to input mode 00
    set pupdr to pull up          01
    */
       //Enable clock GPIOE
       RCC->AHB2ENR |= RCC_AHB2ENR_GPIOEEN;

    //for the output pins PE[10:13]
      //set MODER to output mode       01
       GPIOE->MODER |= GPIO_MODER_MODE10_0; //pin 10 bit 0 want value 1
```

```
      GPIOE->MODER &= ~GPIO_MODER_MODE10_1;//pin 10 bit 1 want value 0
      GPIOE->MODER |= GPIO_MODER_MODE11_0; //pin 11 bit 0 want value 1
      GPIOE->MODER &= ~GPIO_MODER_MODE11_1;// pin 11 bit 1 want value
0
      GPIOE->MODER |= GPIO_MODER_MODE12_0; // pin 12 bit 0 want value
1
      GPIOE->MODER &= ~GPIO_MODER_MODE12_1;// pin 12 bit 1 want value
0
      GPIOE->MODER |= GPIO_MODER_MODE13_0; // pin 13 bit 0 want value
1
      GPIOE->MODER &= ~GPIO_MODER_MODE13_1;// pin 13 bit 1 want value
0

    //set otyper to output open drain
      GPIOE->OTYPER |= GPIO_OTYPER_OT_10;//pin 10 want value 1
      GPIOE->OTYPER |= GPIO_OTYPER_OT_11;//pin 11 want value 1
      GPIOE->OTYPER |= GPIO_OTYPER_OT_12;//pin 12 want value 1
      GPIOE->OTYPER |= GPIO_OTYPER_OT_13;//pin 13 want value 1

    //set pupdr to no pull up/pull down 00

           GPIOE->PUPDR &= ~GPIO_PUPDR_PUPD10_0;//pin 10 bit 0 want
value 0
      GPIOE->PUPDR &= ~GPIO_PUPDR_PUPD10_1;//pin 10 bit 1 want value 0
      GPIOE->PUPDR &= ~GPIO_PUPDR_PUPD11_0;//pin 11 bit 0 want value 0
      GPIOE->PUPDR &= ~GPIO_PUPDR_PUPD11_1;//pin 11 bit 1 want value 0
      GPIOE->PUPDR &= ~GPIO_PUPDR_PUPD12_0;//pin 12 bit 0 want value 0
      GPIOE->PUPDR &= ~GPIO_PUPDR_PUPD12_1;//pin 12 bit 1 want value 0
      GPIOE->PUPDR &= ~GPIO_PUPDR_PUPD13_0;//pin 13 bit 0 want value 0
      GPIOE->PUPDR &= ~GPIO_PUPDR_PUPD13_1;//pin 13 bit 1 want value 0


    //for the input pins PA[0:3]
    //set MODER to input mode 00

      GPIOA->MODER &= ~GPIO_MODER_MODE0_0;//pin 0 bit 0 want value 0
      GPIOA->MODER &= ~GPIO_MODER_MODE0_1;//pin 0 bit 1 want value 0
      GPIOA->MODER &= ~GPIO_MODER_MODE1_0;//pin 1 bit 0 want value 0
      GPIOA->MODER &= ~GPIO_MODER_MODE1_1;//pin 1 bit 1 want value 0
      GPIOA->MODER &= ~GPIO_MODER_MODE2_0;//pin 2 bit 0 want value 0
      GPIOA->MODER &= ~GPIO_MODER_MODE2_1;//pin 2 bit 1 want value 0
      GPIOA->MODER &= ~GPIO_MODER_MODE3_0;//pin 3 bit 0 want value 0
      GPIOA->MODER &= ~GPIO_MODER_MODE3_1;//pin 3 bit 1 want value 0

          //No pull-up no pull-down on GPIOA pins.
```

```
        GPIOA->PUPDR &= 0xFFFFFFF0;


    //row most recently pressed. contains values of 0 to 4 zero means
no rows selected. when button is pressed only store 1:4
    location[0] = 0;
    //col mos recently pressed. contains values of 0 to 4 zero means
no colums selected. when button is pressed only store values from 1 to
4
    location[1] = 0;
    prevColumnResultIDR = 0;
}

void scanKeypad()
{
    /*
    set output pins PE[10:13] ODR to 0000
    read values of input pins PA[0:3]

    if one of the pins is zero and it has not changed do not allow
another press(dont allow long button press to print multiple #s)
    if one of the pins is zero begin the looping through the scan
process

    after scanning wait for an amount of time to debounce
    */
        findFlag = 0;
    GPIOE->ODR = 0x00000000;
        debounce(10000);
    ColumnResultIDR = GPIOA->IDR;
        ColumnResultIDR &= 0x000000FF;

        //Switch to column based on ColumnResult
      switch(ColumnResultIDR){



         case 0x0000000E:
              location[0] = 1;
              findLocation();
                          debounce(100);
              break;

         case 0x0000000D:
              location[0] = 2;
```

```
                findLocation();
                                debounce(100);
                break;

            case 0x0000000B:
                location[0] = 3;
                findLocation();
                            debounce(100);
                break;

            case 0x00000007:
                location[0] = 4;
                findLocation();
                debounce(100);
                                break;
      }
            //display message will pull the data from location[] and
display the correct character in the next slot
           if (findFlag)
                {
            char character = gridToChar();
            displayMessage(character);
                }

            //reset the location array so that it is not re-used
            location[0] = 0;
            location[1] = 0;
      prevColumnResultIDR = ColumnResultIDR;
}



void findLocation(){
/*
    Scan process()
    set output pins PE[10:13] ODR to 0000
    read values of input pins PA[0:3]
    save the index of the zero input pin as col. this is the column
number

    set output pins PE[10:13] ODR to 1000
    read values of input pins PA[0:3]
    if the PE[col] = 0 then the button pressed was 1,col
    display and return
    else
```

```
      keep going

    repeat for each config 0100, 0010, 0001
0000 0000 0000 0000 0011 1000 0000 0000, = 0x00003800
0000 0000 0000 0000 0011 0100 0000 0000, = 0x00003400
0000 0000 0000 0000 0010 1100 0000 0000, = 0x00002C00
0000 0000 0000 0000 0001 1100 0000 0000  = 0x00001C00
0000 0000 0000 0000 0011 1100 0000 0000  = 0x00003C00
    */
int rowTestIDR = 0x0;

           //Set pin 10 of GPIOE ODR to 0, rescan IDR
           GPIOE->ODR &= 0xFFFFFBFF;
    GPIOE->ODR |= 0x00003800;
           debounce(10000);
    rowTestIDR = GPIOA->IDR;
    rowTestIDR &= 0x0000000F;

    //if the IDR reads the same value that it was given originally
after setting the ODR then we have found the correct row
    if(rowTestIDR == ColumnResultIDR)
    {
      location[1] = 1;
            findFlag = 1;
      return;
    }

           //Set pin 11 of GPIOE ODR to 0, rescan IDR
           GPIOE->ODR &= 0xFFFFF7FF;
    GPIOE->ODR |= 0x00003400;
           debounce(10000);
    rowTestIDR = GPIOA->IDR;
    rowTestIDR &= 0x0000000F;
    //if the IDR reads the same value that it was given originally
after setting the ODR then we have found the correct row
    if(rowTestIDR == ColumnResultIDR)
    {
      location[1] = 2;
                findFlag = 1;
      return;
    }

           //Set pin 12 of GPIOE ODR to 0, rescan IDR
           GPIOE->ODR &= 0xFFFFEFFF;
    GPIOE->ODR |= 0x00002C00;
```

```
            debounce(10000);
    rowTestIDR = GPIOA->IDR;
    rowTestIDR &= 0x0000000F;
    //if the IDR reads the same value that it was given originally
after setting the ODR then we have found the correct row
    if(rowTestIDR == ColumnResultIDR)
    {
      location[1] = 3;
                findFlag = 1;
      return;
    }


    //Set pin 13 of GPIOE ODR to 0, rescan IDR
           GPIOE->ODR &= 0xFFFFDFFF;
    GPIOE->ODR |= 0x00001C00;
           debounce(10000);
    rowTestIDR = GPIOA->IDR;
    rowTestIDR &= 0x0000000F;
    //if the IDR reads the same value that it was given originally
after setting the ODR then we have found the correct row
    if(rowTestIDR == ColumnResultIDR)
    {
      location[1] = 4;
                findFlag = 1;
      return;
    }


}

char gridToChar()
{
    if(location[0] == 1)
    {
            switch(location[1]){
                  case 1://1
                        return '1';
                  case 2://4
                  return '4';
                  case 3://7
                        return '7';
                  case 4://*
                  return 'S';
      }
```

```
    }

      if(location[0] == 2)
    {
          switch(location[1]){
               case 1://2
                    return '2';
               case 2://5
               return '5';
               case 3://8
                    return '8';
               case 4://0
               return '0';
      }
    }

      if(location[0] == 3)
    {
          switch(location[1]){
               case 1://3
                    return '3';
               case 2://6
               return '6';
               case 3://9
                    return '9';
               case 4://#
               return 'H';
      }
    }

      if(location[0] == 4)
    {
          switch(location[1]){
               case 1://A
                    return 'A';
               case 2://B
               return 'B';
               case 3://C
                    return 'C';
               case 4://D
               return 'D';
      }
    }
         return 0xFF;
}
```

```
void debounce(int duration)
{
  for (int i =0 ; i < duration; i++)
    {
       (void)0;
    }
}

void displayMessage(char newChar)
{
    if (msgCount < 6)
    {
    message[msgCount] = (uint8_t)newChar;
    uint8_t* lcdOutput;
    lcdOutput = message;
       for (uint8_t i=0; i <= msgCount; i++)
       {
            LCD_WriteChar(lcdOutput, false, false, i);
            lcdOutput++;
       }
            msgCount++;
            debounce(500000);
  }
}
```