Final Project: Digital Ohmmeter

University of Oklahoma

Digital Design Lab

Dr. Erik Petrich

ECE 4273-012

Ged Miller

Terreon Brewster

Spring 2023

Introduction:

This project focuses on designing and implementing a digital ohmmeter with auto-ranging functionality using an LPC1769 microcontroller. The ohmmeter can measure resistance values between 100 ohms and 1 megaohm. The measurement results are displayed on an LCD HD44780 display, providing users with an easy-to-read interface. The system employs two CD4051B multiplexers and a set of known reference resistors to achieve auto-ranging capabilities, ensuring accurate measurements within one percent across the specified range. The has an added cool functionality to output a 7MHz square wave signal to a speaker system.

Functionalities:

1. Core Functionality:
    - Measure resistance between 100 ohms and 10k ohms with auto-ranging capabilities
    - Display measured resistance on the LCD HD44780 display
2. Desired Functionality:
    - Extend the measurement range to 1M ohms
    - Implement a user-friendly interface for displaying results
3. Additional Functionality:
    - Add support for calibration
    - Implement power-saving features when the device is idle
    - Produce a 7Mhz square wave
    - Output to speaker
    - Increment frequency with a button

User Manual:

This program is designed to measure resistance using a set of reference resistors, display the measured resistance on an LCD screen, and allow the user to increment output frequency with the system using a button. Follow the instructions below to use the program:

1. Hardware Setup:
    a. Ensure that the HD44780 LCD screen is properly connected to the microcontroller according to the code's pin assignments.
    b. Connect one leg of the unknown resistor to ground and the other to pin 3 of the CD4051B IC (also known as the COM pin).
    c. Ensure the speaker is connected to PAD 10 of the LPC1769 microcontroller.
    d. Plug in the LPC1769 to your computer using the provided USB cable.

    e. Connect a 5V power source to pin 2 of the HD44780 display and Ground to pin 1 of the HD44780 display.
2. Software Setup:
    a. Launch MCUXpresso IDE on your computer.
    b. Load the FinalProjectCode.c project file into the IDE.
    c. Debug the program and ensure that the build is successful.
3. Running the Program:
    a. Power on the microcontroller.
    b. Run the program in the MCUXpresso IDE.
    c. Wait for the LCD screen to display the initial messages.
4. Resistance Measurement and Speaker Frequency Control:
    a. The resistance of the resistor will automatically be measured and displayed on the HD44780 display.
    b. If you want to measure the resistance of another resistor, simply replace the unknown resistor with a new one.
    c. Press the button (SW1) to increment the frequency of the speaker.
5. Powering Off:
    a. To power off the microcontroller, stop the program execution in the MCUXpresso IDE.
    b. Disconnect the power source from the microcontroller.

Hardware Overview:

1. LPC1769 microcontroller: Responsible for controlling the overall system and making resistance measurements
2. CD4051B analog multiplexer: Used for selecting the appropriate reference resistor depending on the range of the unknown resistor
3. Resistors: Nine reference resistors to cover the desired measurement range
4. LCD HD44780 alphanumeric display: Provides a user-friendly interface for displaying the measured resistance values
5. Capacitors: Used as required for filtering and decoupling purposes
6. Power supply: Provides power to all components

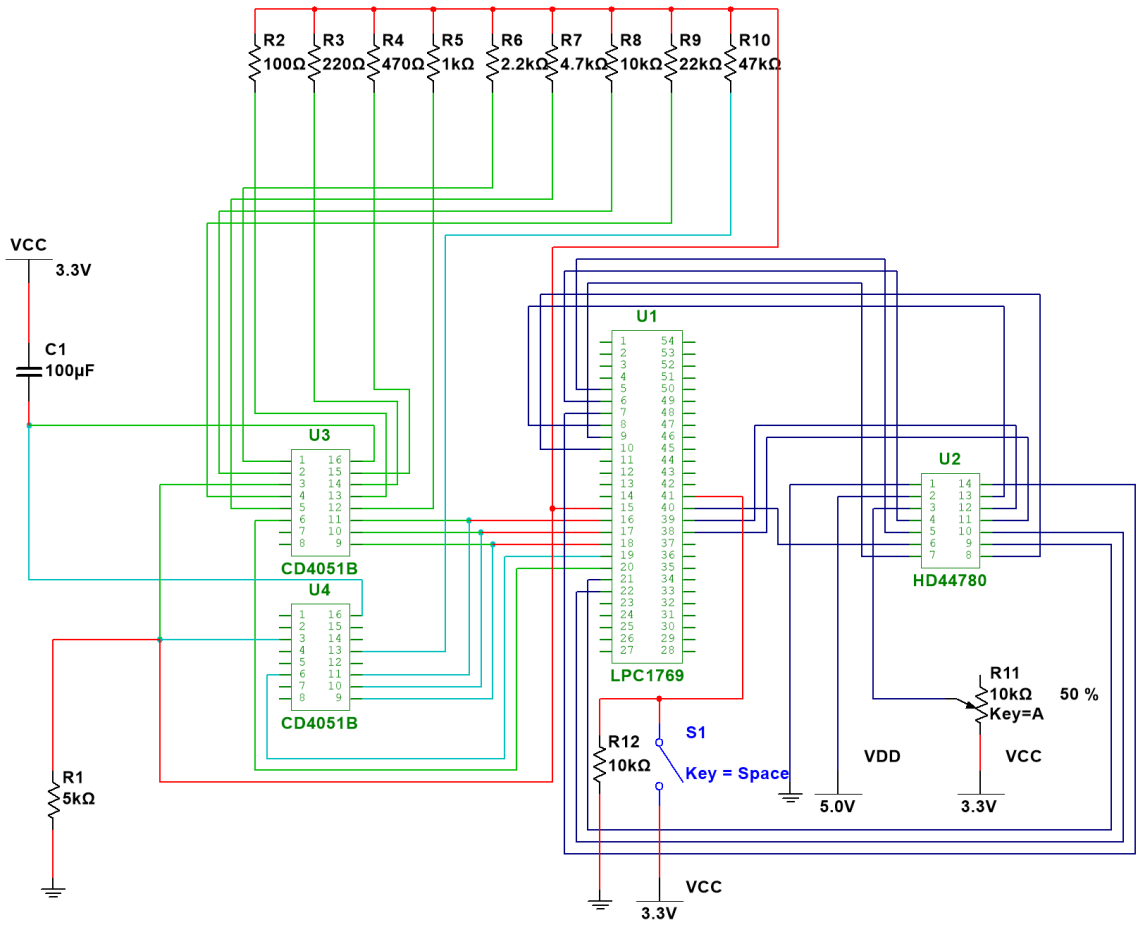

Figure 1: Schematic of LPC1769 auto-ranging ohmmeter and LCD Display

To determine the necessary impedances of the reference resistors, the range ratio must be calculated. The range ratio is the factor by which the range expands for each subsequent reference resistor. To find the range ratio, use the following formula:

$$Range\ Ratio = (Max\ resis\tan c\ e/Min\ resis\tan c\ e)^{(1/(Numberofresistors-1))}$$

$$Range\ Ratio = (1{,}000{,}000/100)^{(1/(9-1))} \approx 2.154$$

Figure 2: Range Ratio calculation.

Then the reference resistor voltages are calculated using the Range Ratio above and allow for auto-ranging to be within one percent of the true value.

$$R1 = 100ohms$$

$$R2 = R1 * Range\ Ratio \approx 100 * 2.154 \approx 220ohms$$

$$R3 = R2 * Range\ Ratio \approx 220 * 2.154 \approx 470ohms$$

$$R4 = R3 * Range\ Ratio \approx 470 * 2.154 \approx 1{,}000ohms$$

$$R5 = R4 * Range\ Ratio \approx 1{,}000 * 2.154 \approx 2{,}200ohms$$

$$R6 = R5 * Range\ Ratio \approx 2{,}200 * 2.154 \approx 4{,}700ohms$$

$$R7 = R6 * Range\ Ratio \approx 4{,}700 * 2.154 \approx 10{,}000ohms$$

$$R8 = R7 * Range\ Ratio \approx 10{,}000 * 2.154 \approx 22{,}000ohms$$

$$R9 = R8 * Range\ Ratio \approx 22{,}000 * 2.154 \approx 47{,}000ohms$$

Figure 3: Reference impedance calculations.

Software Overview:

1. wait(float secs): This function introduces a delay in the execution of the program for a specified number of seconds.
2. LCDwriteCommand(unsigned char num) and LCDwriteData(unsigned char data): These functions write command and data respectively to the LCD display. The LCDwriteCommand function writes a command byte to the display, while the LCDwriteData function writes a data byte.
3. LCDInitialization(void): This function initializes the LCD display by setting the appropriate control signals and sending the necessary commands to configure the display.
4. measure_resistance(void): This function measures the resistance value using the ADC and a set of reference resistors. The function iterates through the reference resistors, calculating the resistance for each and returning the value when it falls within a specific range.
5. moveCursor(void): This function moves the cursor on the LCD display.
6. writeString(char* str): This function writes a string to the LCD display by sending each character in the string as data to the display.
7. feedSeq(void) and setPLL(void): These functions configure the PLL (Phase-Locked Loop) of the microcontroller for system clock generation.
8. getSw1(void): This function reads the state of a switch (Sw1) connected to the microcontroller.
9. main(void): The main function sets up the microcontroller's GPIO pins and initializes the LCD display. It then enters a loop where it reads the resistance value, updates the LCD display, and checks the state of the switch.

Methods:

Hardware:

The LPC1769 acts as an auto-ranging ohmmeter by first connecting the 9 reference resistors as described in Figure 3 to the IN/OUT Channels of the CD4051B IC's. The other leg is then connected to the ADC pin 15 on the LPC1769 microcontroller. The reference resistor is then connected to pin 3 on both of the CD4051B IC's which are both connected to the ADC pin 15 on the LPC1769 via a node.

The LPC1769 interfaces with the HD44780 display via 12 GPIO ports. These ports are set as outputs to the RS, R/W, E, and DB0-DB7 ports on the display. The display pin 1 is connected to Ground and the display pin 2 is connected to an external 5V power source.

The speaker is connected to PAD 10 of the LPC1769 which outputs a square wave signal to the speaker. A switch is connected to pin 50 of the LPC1769 to increment the frequency signal.

Software:

For the software portion of this project, the LPC1769 utilizes the Analog-to-Digital (ADC) and General-Purpose Input/Output (GPIO) subsystems. After successfully initializing the ADC and GPIO, the LPC1769 will then take an analog input, in this case an unknown resistor, and use the 9 reference resistors to accurately measure the resistance of the unknown resistor within a 1% error. Since the unkown resistor is connected in junction with the other leg of the 9 resistors, the CD14051B chips, and pin 15 on the LPC1769, this will cause the CD1405B chips to determine the appropriate known resistor to measure the resistance of the unknown resistor.

Once resistance has been accurately measured, the results of the measurement would appear on the HD44780 LCD through the GPIO subsystem. This is done by setting the GPIO ports as outputs and developing code that would output the correct measurement and statements gathered from the ADC subsystem on the LCD screen, specifically the *LCDwriteCommand, LCDwriteData,* and the *LCDInitialization* functions. These functions utilize ASCII characters to create commands and characters for the HD44780 LCD.

Results:

The system did not achieve the desired goal of displaying an output to the HD44780 display, measuring a reference resistor, auto-ranging the reference resistor to ensure error of less than one percent, or incrementing the 7Mhz output frequency via a button.

Originally the system was to use an additional chip, the MCP23017, and output ABC select pins to the MCP23017 GPIOA and GPIOB to ports to allow for auto-ranging functionality. This, however, required an I2C system. An assumption was made that the MCP23017 and HD44780 was compatible with an SPI system. This assumption was incorrect. An SPI system was created and unable to interface with the MCP23017 or the HD44780, both designed for I2C.

The MCP23017 chip was removed and the select pins of the CD4051B chips were then rerouted to the ADC channel select pin of the LPC1769. Incidentally, the ADC channel system was

unable to be completed in time, thus rendering the ohmmeter functionality of the system inoperable.

New code for the HD44780 was generated to interface directly with the LPC1769 via 12 GPIO pins instead of interfacing with eh MCP23017 as originally intended. Overall, the connection was successful; however, the initialization function for the HD44780 had some errors that have yet to be rectified. In it's current state, the HD44780 displays a moving cursor that patters about the four display rows.

ASCII functionality was not fully implemented and displayed incorrect characters when called. The display functions that would display the number of ohm's was not completed since outputting strings through the writeString() function failed to display the correct ASCII text. A solution for this problem is not yet known.

Lastly, the speaker functionality failed for several reasons. Although the code was sound and did display a 7Mhz square wave, the output current was too low to effectively drive the speaker. Secondly, the code designed to control the button that would increment the frequency would brick the microcontroller. Only after removing all functions related to the button would the microcontroller work as usual. This was an unforeseen issue since the code running the button is similar to code the team used in Assignment two.

Conclusion Overview:

In summary, this project proposal sough to use five features from the provided list: an ohmmeter from 100 to 10kΩ, an ohmmeter auto range extension to 1MΩ, an LCD display interface, an SPI system, and awesome feature. These features did not come to fruition due to incorrect assumptions about the capabilities of the MCP23017 and the HD44780. To improve this project, using the I2C system would allow for implementation project goals. To further improve the design, adding decoupling resistors to reduce noise and adding a user interface keypad would be beneficial to the project design.

Acknowledgements:

Two lab technicians, Computer Engineer Ged Miller and Electrical Engineer Terreon Brewster developed the software and hardware solutions. Advice given by Dr. Erik Petrich and a Teacher's Assistant.

Code Appendix:

```
#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include <cr_section_macros.h>

#include <stdio.h>

#define FIO0DIR (*(volatile unsigned int *)0x2009C000)
#define FIO0PIN (*(volatile unsigned int *)0x2009C014)
#define FIO0SET (*(volatile unsigned int *)0x2009C018)
#define FIO0CLR (*(volatile unsigned int *)0x2009C01C)
#define PINSEL0 (*(volatile unsigned int *)0x4002C000)
#define PINSEL1 (*(volatile unsigned int *)0x4002C004)
#define PCLKSEL0 (*(volatile unsigned int *)0x400FC1A8)
#define PCLKSEL1 (*(volatile unsigned int *)0x400FC1AC)
#define AD0CR (*(volatile unsigned int *)0x40034000)
#define AD0GDR (*(volatile unsigned int *)0x40034004)
#define PCON (*(volatile unsigned int *)0x400FC0C0)
#define PCONP (*(volatile unsigned int *)0x400FC0C4)
#define AD0DR0 (*(volatile unsigned int *)0x40034010)
#define AD0DR2 (*(volatile unsigned int *)0x40034018)

#define FIO2DIR (*(volatile unsigned int *)0x2009c040)
#define FIO2PIN (*(volatile unsigned int *)0x2009c054)

#define CLKSRCSEL (*(volatile unsigned int *)0x400FC10C)

#define PLL0CON (*(volatile unsigned int *)0x400FC080)
#define PLL0CFG (*(volatile unsigned int *)0x400FC084)
#define PLL0STAT (*(volatile unsigned int *)0x400FC088)
#define PLL0FEED (*(volatile unsigned int *)0x400FC08C)

#define PLL1CON (*(volatile unsigned int *)0x400FC0A0)
```

```
#define PLL1CFG (*(volatile unsigned int *)0x400FC0A4)
#define PLL1STAT (*(volatile unsigned int *)0x400FC0A8)
#define PLL1FEED (*(volatile unsigned int *)0x400FC0AC)

#define CCLKCFG (*(volatile unsigned int *)0x400FC104)
#define USBCLKCFG (*(volatile unsigned int *)0x400FC108)
#define PCLKSEL0 (*(volatile unsigned int *)0x400FC1A8)
#define PCLKSEL1 (*(volatile unsigned int *)0x400FC1AC)

#define AD0CR (*(volatile unsigned int *)0x40034000)
#define AD0GDR (*(volatile unsigned int *)0x40034004)
#define AD0DR0 (*(volatile unsigned int *)0x40034010)
#define AD0DR2 (*(volatile unsigned int *)0x40034018)
#define PCON (*(volatile unsigned int *)0x400FC0C0)
#define CLKOUTCFG (*(volatile unsigned int *)0x400FC1C8)
#define PINSEL3 (*(volatile unsigned int*)0x4002c00c)

//volatile int number;

void wait(float secs) // Wait function.
{
   volatile float count = secs*21.33e6;
   while (count>0)
      count--;
}

void LCDwriteCommand(unsigned char num){

FIO0CLR = (1<<8); //Pin P0.0 as RS, set to low
FIO0CLR = (1<<9); //Pin P0.1 as R/W, set to low
FIO0SET = (1<<10); //Pin P0.2 as E, set to high



const int portbit[] = {0, 1, 2, 3, 4, 5, 6, 7};

for (int x=0; x < sizeof(portbit) / sizeof(int); x++) {
if((num >> x) & 1) {
FIO0PIN |= (1 << portbit[x]);
} else {
```

```
  FIO0PIN &= ~(1 << portbit[x]);
}
}


wait(0.004); // Wait 4ms
FIO0CLR = (1<<10); // Set E Low
}

void LCDwriteData(unsigned char data){

FIO0SET = (1 << 8); // Set RS high for data
FIO0CLR = (1 << 9); // Set R/W low for write
FIO0SET = (1 << 10); // Set E high

   // Write data to the data pins
   for (int i = 0; i < 8; i++) {
      if (data & (1 << i)) {
         FIO0SET = (1 << (i + 6)); // Set data bit i
      } else {
         FIO0CLR = (1 << (i + 6)); // Clear data bit i
      }
   }

   // Toggle enable
   wait(0.00001);
   FIO0CLR = (1 << 10); // Set E low
   wait(0.00001);


}

void LCDInitialization(void){

FIO0CLR = (1<<8); //Pin P0.0 as RS, set to low
FIO0CLR = (1<<9); //Pin P0.1 as R/W, set to low
FIO0CLR = (1<<10); //Pin P0.2 as E, set to low

wait(0.004); // Wait 4ms
```

```
FIO0CLR = (1<<9); //Drive R/W to Ground
//FIO0SET = (1<<0); //Drive RS high
FIO0SET = (1<<10); //Drive E high
wait(0.001);
FIO0CLR = (1<<10); //Drive E low
wait(0.00001);
FIO0SET = (1<<8); //Drive RS high

//0x38 which selects 8 bit bus 00111000

FIO0CLR = (1<<0);
FIO0CLR = (1<<1);
FIO0CLR = (1<<2);
FIO0SET = (1<<3);
FIO0SET = (1<<4);
FIO0SET = (1<<5);
FIO0CLR = (1<<6);
FIO0CLR = (1<<7);

FIO0CLR = (1<<8); //Drive RS low

FIO0CLR = (1<<9); //Drive R/W to Ground
//FIO0SET = (1<<0); //Drive RS high
FIO0SET = (1<<10); //Drive E high
wait(0.001);
FIO0CLR = (1<<10); //Drive E low
wait(0.00001);
FIO0SET = (1<<8); //Drive RS high

//0x06 which selects cursor and cursor direction

FIO0CLR = (1<<0);
FIO0SET = (1<<1);
FIO0SET = (1<<2);
FIO0CLR = (1<<3);
FIO0CLR = (1<<4);
FIO0CLR = (1<<5);
FIO0CLR = (1<<6);
```

```
FIO0CLR = (1<<7);

FIO0CLR = (1<<8); //Drive RS low

FIO0CLR = (1<<9); //Drive R/W to Ground
//FIO0SET = (1<<0); //Drive RS high
FIO0SET = (1<<10); //Drive E high
wait(0.001);
FIO0CLR = (1<<10); //Drive E low
wait(0.00001);
FIO0SET = (1<<8); //Drive RS high

//0x0e makes cursor and display on

FIO0CLR = (1<<0);
FIO0SET = (1<<1);
FIO0SET = (1<<2);
FIO0SET = (1<<3);
FIO0CLR = (1<<4);
FIO0CLR = (1<<5);
FIO0CLR = (1<<6);
FIO0CLR = (1<<7);

FIO0CLR = (1<<8); //Drive RS low

FIO0CLR = (1<<9); //Drive R/W to Ground
//FIO0SET = (1<<0); //Drive RS high
FIO0SET = (1<<10); //Drive E high
wait(0.001);
FIO0CLR = (1<<10); //Drive E low
wait(0.00001);
FIO0SET = (1<<8); //Drive RS high

//0x01 clears display and moves cursor to upper left

FIO0SET = (1<<0);
FIO0CLR = (1<<1);
FIO0CLR = (1<<2);
FIO0CLR = (1<<3);
```

```
FIO0CLR = (1<<4);
FIO0CLR = (1<<5);
FIO0CLR = (1<<6);
FIO0CLR = (1<<7);

FIO0CLR = (1<<8); //Drive RS low

//FIO0CLR = (1<<1); //Drive R/W to Ground
//FIO0SET = (1<<0); //Drive RS high
//FIO0SET = (1<<2); //Drive E high
//wait(0.001);
//FIO0CLR = (1<<2); //Drive E low
//wait(0.00001);

wait(0.004); // Wait 4ms

}



float measure_resistance(void) {
      //reference resistor values in ohms
      float reference_resistors[9] = {100, 220, 470, 1e3, 2.2e3, 4.7e3, 10e3, 22e3, 47e3};
      float voltage;
      float resistance;

      for (int i = 0; i < 9; i++) {
               //set_mux(0, i);

         voltage = 3.3f * adc_read(0) / 4095.0f;
         resistance = reference_resistors[i] * (3.3f - voltage) / voltage;

         if (resistance >= 0.9f * reference_resistors[i] && resistance <= 1.1f * reference_resistors[i]) {
            return resistance;
         }
      }

      return -1.0f; // Indicate an error if the resistance is out of range
   }
```

```
void moveCursor(void){

FIO0CLR = (1<<9); //Drive R/W to Ground
//FIO0SET = (1<<8); //Drive RS high
FIO0SET = (1<<10); //Drive E high

//0x01 clears display and moves cursor to upper left

FIO0CLR = (1<<0);
FIO0CLR = (1<<1);
FIO0CLR = (1<<2);
FIO0CLR = (1<<3);
FIO0CLR = (1<<4);
FIO0CLR = (1<<5);
FIO0CLR = (1<<6);
FIO0SET = (1<<7);

wait(0.001);
FIO0CLR = (1<<10); //Drive E low
wait(0.00001);

}

void writeString(char* str) {
   while (*str) {
      LCDwriteData(*str++);
      wait(0.01); // Add a small delay between characters
   }
}

void feedSeq(void) //Updates PLL0FEED for changes to take effect.
{
PLL0FEED = 0xAA;
PLL0FEED = 0x55;
}

void setPLL(void){
```

```
PLL0CON &= ~(1 << 1); //Step 1: Disconnect PLL

feedSeq(); //Call Feed Sequence

PLL0CON &= ~(1 << 0); //Step Two: Disables PLL, much better than using a million counters.

feedSeq();

CCLKCFG = 0; //Step 3: Using Internal RC

CLKSRCSEL = 0; //Step 4

int M = 35; // From the formula given in the manual to calculate M

PLL0CFG = 34; //Step 5: M - 1

feedSeq();

PLL0CON |= (1 << 0); //Step 6: Enables PLL0

feedSeq();

while((PLL0STAT & (1 << 26)) == 0){} //Step 8: Ensures PLL0STAT is locked by monitoring PLOCK0 bit

CCLKCFG = 39; //Step 7

PLL0CON |= (1 << 1);  //Step 9

feedSeq();




}

int getSw1(void)
{
   return(FIO2PIN >> 8) & 1;
}
```

```
int main(void) {

//GPIO Definitions for LCD Initialization

PINSEL0 &= ~(1<<0); //Configure GPIO Pin DB0
PINSEL0 &= ~(1<<1); //Configure GPIO Pin DB0
PINSEL0 &= ~(1<<2); //Configure GPIO Pin DB1
PINSEL0 &= ~(1<<3); //Configure GPIO Pin DB1
PINSEL0 &= ~(1<<4); //Configure GPIO Pin DB2
PINSEL0 &= ~(1<<5); //Configure GPIO Pin DB2
PINSEL0 &= ~(1<<6); //Configure GPIO Pin DB3
PINSEL0 &= ~(1<<7); //Configure GPIO Pin DB3
PINSEL0 &= ~(1<<8); //Configure GPIO Pin DB4
PINSEL0 &= ~(1<<9); //Configure GPIO Pin DB4
PINSEL0 &= ~(1<<10); //Configure GPIO Pin DB5
PINSEL0 &= ~(1<<11); //Configure GPIO Pin DB5
PINSEL0 &= ~(1<<12); //Configure GPIO Pin DB6
PINSEL0 &= ~(1<<13); //Configure GPIO Pin DB6
PINSEL0 &= ~(1<<14); //Configure GPIO Pin DB7
PINSEL0 &= ~(1<<15); //Configure GPIO Pin DB7
PINSEL0 &= ~(1<<16); //Configure GPIO Pin RS
PINSEL0 &= ~(1<<17); //Configure GPIO Pin RS
PINSEL0 &= ~(1<<18); //Configure GPIO Pin R/W
PINSEL0 &= ~(1<<19); //Configure GPIO Pin R/W
PINSEL0 &= ~(1<<20); //Configure GPIO Pin E
PINSEL0 &= ~(1<<21); //Configure GPIO Pin E
PINSEL0 &= ~(1<<22); //Extra
PINSEL0 &= ~(1<<23); //Extra

FIO0DIR |= (1<<0); //Configure GPIO Pin as output RS
FIO0DIR |= (1<<1); //Configure GPIO Pin as output RS
FIO0DIR |= (1<<2); //Configure GPIO Pin as output R/W
FIO0DIR |= (1<<3); //Configure GPIO Pin as output R/W
FIO0DIR |= (1<<4); //Configure GPIO Pin as output E
FIO0DIR |= (1<<5); //Configure GPIO Pin as output E
FIO0DIR |= (1<<6); //Configure GPIO Pin as output DB0
FIO0DIR |= (1<<7); //Configure GPIO Pin as output DB0
```

```
FIO0DIR |= (1<<8); //Configure GPIO Pin as output DB1
FIO0DIR |= (1<<9); //Configure GPIO Pin as output DB1
FIO0DIR |= (1<<10); //Configure GPIO Pin as output DB2
FIO0DIR |= (1<<11); //Configure GPIO Pin as output DB2
FIO0DIR |= (1<<12); //Configure GPIO Pin as output DB3
FIO0DIR |= (1<<13); //Configure GPIO Pin as output DB3
FIO0DIR |= (1<<14); //Configure GPIO Pin as output DB4
FIO0DIR |= (1<<15); //Configure GPIO Pin as output DB4
FIO0DIR |= (1<<16); //Configure GPIO Pin as output DB5
FIO0DIR |= (1<<17); //Configure GPIO Pin as output DB5
FIO0DIR |= (1<<18); //Configure GPIO Pin as output DB6
FIO0DIR |= (1<<19); //Configure GPIO Pin as output DB6
FIO0DIR |= (1<<20); //Configure GPIO Pin as output DB7
FIO0DIR |= (1<<21); //Configure GPIO Pin as output DB7
FIO0DIR |= (1<<22); //Configure GPIO Pin as output
FIO0DIR |= (1<<23); //Configure GPIO Pin as output

//FIO2DIR &= ~(1<<8); // sw1



PINSEL3 &= ~(1 << 23);
PINSEL3 |= (1 << 22);
CLKOUTCFG = 0;
CLKOUTCFG |= (1 << 8);
setPLL();

AD0CR = (1<<21); // A/D is ON

AD0CR &= ~(1<<16); //disables burst mode

AD0CR |= (1<<0); // selects P0.23 AD0.2 to be read from

while(1){
LCDInitialization();
//moveCursor();
wait(0.001);
LCDwriteCommand(1); //Sets cursor in top left corner
LCDwriteCommand(14); //Turns display and cursor on
```

```
LCDwriteCommand(80); //Set cursor postion


//if(getSw1()){
//                                                  number++;
//                                                  setPLL();
//                                        }


}
}
```