ECE 2036 Lab 2 – Setup and Test mbed I/O Hardware

Check-Off Deadline: Section A – Thursday Feb 5
Section B – Friday Feb 6

Name:_____

Item	TA Signoff
Part 1. (40%) Color LCD Hello World	
Part 2. (10%) Timer display on Color LCD	
Part 3. (25%) Temperature Sensor display on Color LCD	
Part 4. (25%) Four Pushbuttons moving Robot with Sound	
All Checkoffs complete at least two days before deadline +3%	
LCD Robot Animation and Artistic Merit (up to TA +5% max)	

Embedded devices account for 98% of the world's microprocessors. For every desktop computer, there are over 100 embedded devices. A high-end car can contain over 200 microprocessors. Embedded devices contain a computer with software that is typically not changed by the user (called Firmware). Most users are not aware that their cell phones, cameras, audio players, appliances, and TVs contain a computer with firmware. C/C++ is currently the most widely used language for embedded devices. ARM 32-bit RISC processors, which are similar to the one found in the *mbed* module, are used in about 80% of embedded devices including 95% of cell phones. There are over 30 billion ARM processors in the world! There are two ARM processors on your mbed module, one is just used for the USB interface to the main processor. The color LCD has another processor.

This lab will hookup and demo new I/O hardware for your mbed ARM processor module in preparation for lab 3. This assignment will start with getting your LCD display working with the *mbed* board. In addition to your mbed module, you will need your breadboard, wire kit, and color LCD module.

Color Graphics LCD Module

CAUTION

Power on pin 1 (+5V) on the Color LCD in your parts kit **uses only the 5V mbed pin VU** (i.e., **not 3.3V – mbed pin Vout**) and pin 7 (GND) is connected to the mbed GND pin. Always double check power pin connections before turning on power for the first time – if you get them wrong it might **burn out** the device! Be sure to read the Color LCD wiki page for additional help using the Color LCD with mbed and photos. If you want to plug the LCD directly in a breadboard, it is necessary to carefully bend over one pin as shown on the wiki page. The LCD requires several wires and if it is placed near the mbed pins used, the breadboard setup will be easier. Note that the mbed pins used (27, 28, 29) are on the right side of the mbed module and these pins are a bit different than the wiki page example (LCD code can use any of mbed's three Serial outputs just by changing the class constructor pin number arguments), but use these pins to be compatible with the embedded systems you will build in labs that follow to have space for all of the parts added later.

Instructions for Part 1: (40%)

1. Watch this Youtube video FIRST. This tells you how **NOT to break the pins** on your mBED board! **VERY IMPORTANT** - COULD WIND UP COSTING YOU $50!

Video link: http://mbed.org/blog/entry/104/

2. On your protoboard, please put the LCD on the side of the MBED chip where you have the pins p27-29. See the wiring table table that follows for the pin connections needed.

LCD Wiring for LCD test program

Mbed	uLCD pin
5V=VU	5V (1)
Gnd	Gnd (7)
TX=P28	RX (5)
RX=P27	TX (3)
P29	Reset (9)
Note: Bend over LCD pin 10, or use LCD's cable set per the wiki instructions	

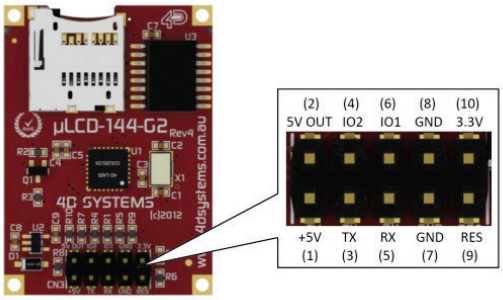

Figure 1: These are the LCD pin connections needed for this assignment. Please note that there are some pins moved between this figure and the mbed cookbook wiki demo code. Note that if you choose to use the cable, the RX and TX pins are already swapped on the cable connector label (per LCD wiki page wiring table and photos).

3. Create your *mbed* user account using the instructions in your mbed box. After placing the *mbed* board on the protoboard, you can connect it to your PC's USB port and this is needed first to setup a new account. It should show up on your computer as an external drive (just like a USB flash drive). You will put **.bin* files in this drive that will be automatically be downloaded to the flash drive and run on your *mbed* board.

4. To quickly get all of the include files needed later, after logging onto mbed.org, "Import" the skeleton code into your *mbed* account's project space that is found at

http://mbed.org/users/4180_1/code/mythermostat/

5. Next, create a new program in your compiler environment and type in the following code. Remember … you will need to copy uLCD_4DGL.h from your skeleton code project into this new project for the include to work. Cut and paste can be used in the compiler's left column that shows project source files. You can also use the import library link found on the LCD wiki page.

```
// Hello World! for the TextLCD
#include "mbed.h"
#include "uLCD_4DGL.h"

uLCD_4DGL uLCD(p28, p27, p29); // create a global lcd object

int main() {
  uLCD.printf("\nHello World!\nyourname");
}
```

6. Replace *"yourname"* above with your name. Compile this program with no erros and a **.bin* file is created to run the code. On some machines, this will automatically download into your *mbed* drive if it is connected to your computer; however, on other machines or web browsers the file might be in your download folder. In this case, all you need to do is drag it to the *mbed* drive and it will be automatically downloaded to the board.

7. Pressing the reset button on the mbed will automatically run the program *(*.bin* file) with the most recent timestamp that is downloaded to the board. Do this and you should see your first message on the LCD display! (assuming every single jumper wire is correctly hooked up – swapping RX and TX is a common error that will prevent writing anything to the LCD). If they are swapped, after the LCD times out with no data, a splash screen appears.

Instructions for Part 2: (10%)

Please use the wiki page found at **https://mbed.org/users/4180_1/notebook/ulcd-144-g2-128-by-128-color-lcd/ and make a timer to continuously display the elapsed time in** seconds underneath your "Hello World!" message. The uLCD.locate(x,y) member function is a handy way to move the text cursor on the LCD before rewriting the time. The mbed timer API is the easy way to start and read the timer. See http://www.cplusplus.com/reference/cstdio/printf/ for printf syntax and options.

Using the Temperature Sensor

Part 3: Digital Thermometer (25%)

An TMP36 wiki page is provided that shows how to connect the analog TMP36 sensor (to p15) and read the temperature using C/C++ on the mbed module. BE CAREFUL because it looks just like the 2N3904 transistor in the kit, so check the tiny gray marks on the case's flat area for the part number. On the wiki page sample code, note the use of C++ classes and operator overloading to read the temperature. This assignment will verify your hardware connections and sensor operation before trying your next mbed lab in a couple of weeks. Make sure the sensor is **not upside down** (per wiki) before connecting power the first time or you may **burn it out**!

Some mbeds will occasionally see a bit of noise on the temperature sensor readings. According to the data sheet, this can happen on analog data measurements when the ARM processor is in debug mode as in the mbed module setup. A bit of noise on the PC's USB 5V power might also be part of the problem. Averaging the reading several times in software and/or adding a small decoupling capacitor to filter the power supply noise and A/D input signal can help alleviate this issue, if it causes problems later. The TAs have some extra capacitors available in the lab.

Instructions For Part 3

1. In your last assignment you got your LCD working, please leave this hooked up! For a visual arrangement for your components, use something like the arrangement at the following YouTube site:

http://www.youtube.com/watch?v=oozKIHzeoQU

Some other parts are added in later labs on the right side of the mbed, so leave space for them.

Note this video has the older BW text LCD, but you want the new color LCD in about the same location to simplify breadboard jumper wiring.
2. Now hook up the temperature sensor using the information on the wiki page at:

http://mbed.org/users/4180_1/notebook/lm61-analog-temperature-sensor/

3. Create a new program and type in or cut and paste the following code in main.cpp that will display the temperature reading from the sensor on the LCD.

```
#include "mbed.h"
#include "uLCD_4DGL.h"
//Print temperature from TMP36 analog temperature sensor
//Setup a new class for TMP36 sensor
class TMP36
{
public:
    TMP36(PinName pin);
    TMP36();
    operator float ();
    float read();
private:
//class sets up the AnalogIn pin
    AnalogIn _pin;
};
TMP36::TMP36(PinName pin) : _pin(pin)
{
// _pin(pin) means pass pin to the AnalogIn constructor
}
float TMP36::read()
{
//convert sensor reading to temperature in degrees C
    return ((_pin.read()*3.3)-0.500)*100.0;
}

//overload of float conversion (avoids needing to type .read() in equations)
TMP36::operator float ()
{
//convert sensor reading to temperature in degrees C
    return ((_pin.read()*3.3)-0.500)*100.0;
}

//use the new class to set p15 to analog input to read and convert TMP36 sensor's
voltage output
TMP36 myTMP36(p15);

//also setting unused analog input pins to digital outputs reduces A/D noise
//see http://mbed.org/users/chris/notebook/Getting-best-ADC-performance/
DigitalOut P16(p16);
DigitalOut P17(p17);
DigitalOut P18(p18);
DigitalOut P19(p19);
DigitalOut P20(p20);

uLCD_4DGL uLCD(p28, p27, p29); // create a global lcd object

int main()
{
    float tempC, tempF;

    while(1) {
        tempC = myTMP36.read();
        //convert to degrees F
        tempF = (9.0*tempC)/5.0 + 32.0;
        //print current temp
        uLCD.locate(2,2);
        uLCD.printf("%5.2F C %5.2F F \n\r", tempC, tempF);
        wait(.5);
    }
}
```

Part 4: Four Button Keyboard & Robot (25%)

In the last part of this assignment you will build a robot display on the LCD that moves using four pushbuttons (F, B, L, R). Do not take off the other components! The point of this exercise is to build up and verify your components work to help with your future mbed lab.

Using Pushbuttons

Read the pushbutton wiki page and watch the videos for additional help using pushbuttons with mbed. Small pushbuttons in the kit are available for use on your breadboard. Look at the constructors in the sample code to determine the pin connections to the mbed board. Note that the pushbutton needs to be plugged in just like an IC (i.e., note flat part of pins on sides) to avoid shorting the switch.

Using the Speaker

The driver transistor and speaker are used to make different tones when each button is pressed and the robot moves. See the speaker wiki page for additional hardware and software help using speakers with mbed. Look at the constructors in the sample code that follows to determine the pin connections to the mbed board (i.e., p21). The jumper wires will need to be connected underneath the speaker before it is plugged into the breadboard. It is also possible to place it with the pins a bit diagonally oriented so that the speaker case does not cover the breadboard pins needed to hook it up. This makes it easier to use the flexible jumpers wire in the new kit.

Instructions For Part 4

1. Please leave all previous components hooked up on your board (i.e. LCD and temperature sensor) – It will save you time later!

2. Now hook up four push buttons to pins 23-26 using the information on the wiki page at:

http://mbed.org/users/4180_1/notebook/pushbuttons/

3. Now hook up the speaker and 2N3904 transistor driver to pin 21 using the information on the wiki page found at:

https://mbed.org/users/4180_1/notebook/using-a-speaker-for-audio-output/

4. After logging in at mbed.org, use this import program link:
http://developer.mbed.org/users/4180_1/code/my2036lab2robotLCD/
Then click the "Import this Program" button to import the entire project along with all of needed files into the compiler (This saves typing the code below having to add the *Speaker, PinDetect, uLCD_4DGL, and Robot.h* include (*.h) source code files to the new project). A compiler window opens automatically to import the code.

```
#include "mbed.h"
#include "Speaker.h"
#include "PinDetect.h"
#include "uLCD_4DGL.h"
// setup builtin LEDs
DigitalOut myLed1(LED1);
DigitalOut myLed2(LED2);
DigitalOut myLed3(LED3);
DigitalOut myLed4(LED4);
// setup pushbuttons for debounced interrupt use
PinDetect pb1(p23);
PinDetect pb2(p24);
PinDetect pb3(p25);
PinDetect pb4(p26);

// setup LCD
uLCD_4DGL uLCD(p28, p27, p29); // create a global lcd object

//moving include down here allows use in uLCD in Robot class draw and erase member functions
//this is easier for now than passing a new uLCD_4DGL object argument to the class
#include "Robot.h"

// setup instance of new Speaker class, mySpeaker using pin 21
// the pin must be a PWM output pin
Speaker mySpeaker(p21);

// pushbutton status - need volatile keyword for safe interrupt R/W
volatile int pbStatus = 0;

// Callback routine is interrupt activated by a debounced pb1 hit
void pb1_hit_callback (void)
{
   // CODE HERE WILL RUN WHEN INTERUPT IS GENERATED
   pbStatus = pbStatus | 0x01;

}
// Callback routine is interrupt activated by a debounced pb2 hit
void pb2_hit_callback (void)
{
   // CODE HERE WILL RUN WHEN INTERUPT IS GENERATED
   pbStatus = pbStatus | 0x02;
}
// Callback routine is interrupt activated by a debounced pb3 hit
void pb3_hit_callback (void)
{
```

```
   // CODE HERE WILL RUN WHEN INTERUPT IS GENERATED
   pbStatus = pbStatus | 0x04;
}
// Callback routine is interrupt activated by a debounced pb4 hit
void pb4_hit_callback (void)
{
   // CODE HERE WILL RUN WHEN INTERUPT IS GENERATED
   pbStatus = pbStatus | 0x08;
}
// main program
int main()
{
   uLCD.printf("\nmyRobot!\nyourname");
//setup four SPST push buttons
   pb1.mode(PullUp); //add internal pullup resistor
   pb2.mode(PullUp);
   pb3.mode(PullUp);
   pb4.mode(PullUp);
   // need a small delay for initial pullup to take effect due to capacitance
   wait(.01);
   // Setup Interrupt callback functions for each pb hit
   pb1.attach_deasserted(&pb1_hit_callback);
   pb2.attach_deasserted(&pb2_hit_callback);
   pb3.attach_deasserted(&pb3_hit_callback);
   pb4.attach_deasserted(&pb4_hit_callback);
   // Start sampling the pb inputs using interrupts
   pb1.setSampleFrequency();
   pb2.setSampleFrequency();
   pb3.setSampleFrequency();
   pb4.setSampleFrequency();
// pushbuttons are all now setup with pullups and running with a software debounce filter

// use your modified robot class to move and display robot's new location on LCD
   Robot myRobot;

   myRobot.draw(); //initial robot display in center of LCD
//main loop to control and display robot
   while(1) {
      //check pushbutton status for new input
      switch (pbStatus) {
            //move forward
         case 0x01 :
            myLed1 = 1;
            pbStatus = 0;
            mySpeaker.PlayNote(200.0,0.1,0.8);
            myRobot.erase();
            myRobot.moveForward(1);
            myRobot.draw();
            break;
```

```
//move backwards
        case 0x02 :
           myLed2 = 1;
           pbStatus = 0;
           mySpeaker.PlayNote(400.0,0.1,0.8);
           myRobot.erase();
           myRobot.moveBackward(1);
           myRobot.draw();
           break;
         //move left
        case 0x04 :
           myLed3 = 1;
           pbStatus = 0;
           mySpeaker.PlayNote(600.0,0.1,0.8);
           myRobot.erase();
           myRobot.moveLeft(1);
           myRobot.draw();
           break;
           //move right
        case 0x08 :
           myLed4 = 1;
           pbStatus = 0;
           mySpeaker.PlayNote(800.0,0.1,0.8);
           myRobot.erase();
           myRobot.moveRight(1);
           myRobot.draw();
           break;
           //no pb or multiple pbs hit
        default :
//            myLed1 = 0;
           myLed2 = 0;
           myLed3 = 0;
           myLed4 = 0;
           pbStatus = 0;
     } //end switch
     myLed1 = !myLed1;
     wait(.05);
  } //end while
} //end main
```

5. This program will allow you to test the LCD, pushbutton, and speaker hookups without changes. The LCD should display a hello world type message and hitting each pushbutton should beep and light up a different LED. Check that this works before going to the next step to display the robot image. LED 1 flashes to indicate the while loop is still running, but it will change just a bit whenever pb1 is hit.

6. Use your Robot class code from lab 1 to provide the missing class and new graphics code in the *Robot.h* include file. The *Robot.h* file in the project provides functions so that it compiles, but with no code inside them (you must add it). To display the robot, some new features are added to the class. Main already calls these functions to display and move the robot, so once they are added the robot should appear and move around correctly on the LCD when pushbuttons are hit.

Add a default constructor to set both xPosition and yPosition to 63 (so robot starts out in the center of the LCD pixel area - 0.0 is upper left corner of LCD).

Add two new member functions to draw and erase the robot called erase() and draw(). Draw should use the existing uLCD class member functions (see LCD wiki page's example code) to draw a robot in a 5 by 5 pixel area with its center provided by the x and y values maintained inside the class. Some handy functions are filled circle, filled rectangle, and even a single pixel. Erase draws over the old robot 5 by 5 pixel area in black to erase it. A black rectangle is an easy way to do erase. As the robot moves, the code erases the old robot first, moves it, and then draws it at the new location.

The robot can be as simple as just a small colored square or circle, but for the optional LCD artistic merit points it needs to look a bit more like a robot and could even have some animation.