I^2C Communication with an Arduino

Alex Lange

ECE 480 – Design Team 3

November 13, 2015

Executive Summary:
Arduino is an open-source microcontroller perfect for prototyping or hobbyists. They are easy to use, have a multitude of optional Arduino-compatible boards such as GPS and Ethernet, and have a vast user base that can help support any project. For these reasons, as well as the ability to use I^2C, Arduino microcontrollers are a quality option for any project.

Keywords:
Arduino, IIC, I2C

Objective:

This application note will serve as a basis for using I^2C to communicate with many different devices, including but not limited to LCDs, sensors, digital-to-analog converters. This application note will focus on using I^2C to display information on multiple LCDs. This application note also assumes a basic understanding of programming as well some familiarity with Arduino.

Introduction:

When using an Arduino for any project, one of the main areas of concern is the limited inputs and outputs (I/O). For the Arduino used in this application note, there are 13 digital I/O and 6 analog I/O. When connecting one LCD using normal communication methods, the LCD requires 6 digital pins, leaving only 7 for other desired functions. Using I^2C, it is possible to use multiple LCDs, digital-to-analog converters, and a multitude of sensors with only two total digital pins.

Basics of I^2C Communication:

I^2C is a multi-master, multi-slave, serial bus invented by Philips Semiconductor. The benefits of I^2C are that it only requires two lines per bus; one clock and one data for all the connected devices. The downside of I^2C compared to other communication methods is that it has a slower communication speed. In terms of LCDs, this slower communication speed will have no effect. Figure 1 illustrates the layout of I^2C with a master microcontroller that can send signals to the three slave nodes. The Arduino has its own internal pull-up resistors, so R_p in Figure 1 is not needed when using an Arduino. SDA is serial data line, while SCL is the serial clock line.

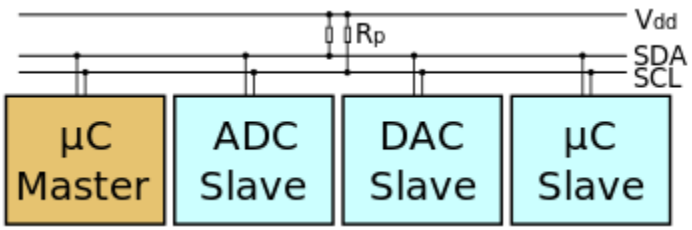


Figure 1: Example I^2C Schematic

Interfacing with External Hardware:

The most common form of LCD used for Arduinos is a 1602 LCD, which is a 16 character by 2 line display. There are a variety of text and background colors that can be selected. The typical setup for connecting one of these LCDs to an Arduino without I^2C can be seen in Figure 2. This demonstrates the complexity associated with standard communication systems.

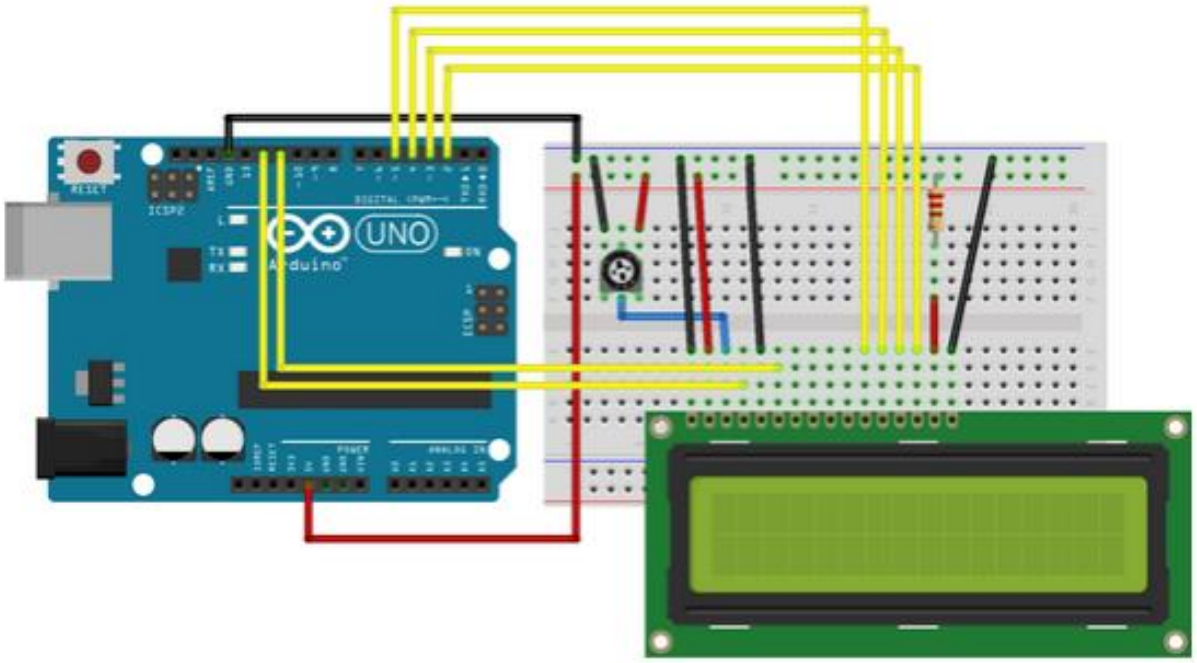


Figure 2: Standard LCD Connection

In order to make the LCD I^2C compatible, an I^2C hardware module is simply soldered to the 16 pins on the LCD. The module selected is very important as there are a vast number of I^2C modules for 1602 LCDs. It is important to get modules with different addresses. If the LCDs have the same address, there is no way for the Arduino to distinguish the two of them. The I^2C module in Figure 3 has the option to change the address. The pads A0, A1, and A2 act as a binary counter. For example, shorting the pads of A0 gives it a value of '1', while leaving it open gives it a '0'. Using this logic, there are 2^3, or 8, possible addresses. Ultimately this means that only 8 LCDs can be displayed if only this model is utilized.

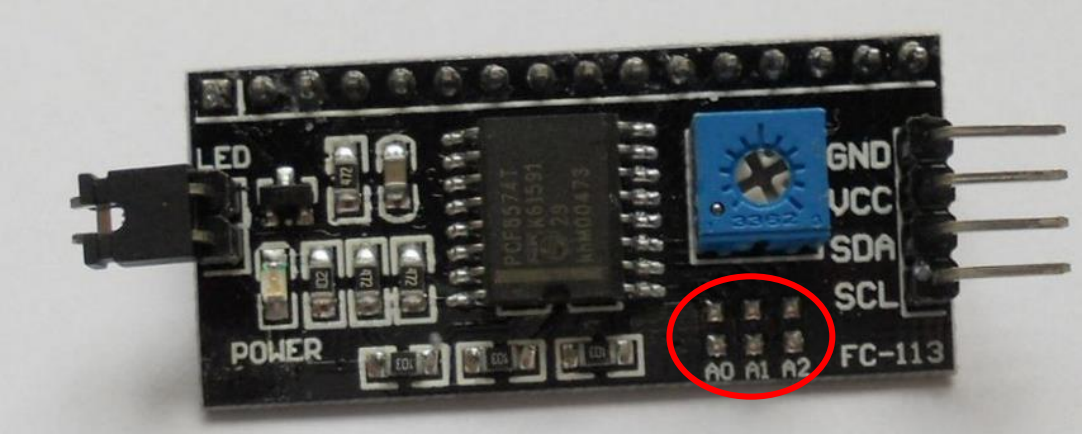


Figure 3: I²C Module for 1602 LCD

After the I²C modules are soldered to the LCDs, the pins from the modules can be connected to those on the Arduino. There are corresponding GND, VCC, SDA, and SCL pins on the Arduino.

Software:

I²C communication can be illustrated by the following analogy. The microcontroller acts as a professor in a classroom talking to students. The classroom is the data bus, where any information on it can be heard by any of the devices, or students. The students act as the devices, only responding when the professor addresses them by their name. The first step to using the I²C devices is to identify the addressable values of each device. These addresses are how the Arduino can communicate with each device individually. This can be done simply by running the following I²C scanner, Figure 4, where the address of each device will be displayed in the Arduino integrated development environment (IDE) serial monitor.

```
#include <Wire.h>

void setup()
{
  Wire.begin();
  Serial.begin(9600);
  Serial.println("\nI2C Scanner");
}


void loop()
{
  byte error, address;
  int nDevices;
  Serial.println("Scanning...");
  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();
    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address,HEX);
      Serial.println("  !");
      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknow error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");
  delay(5000);
}
```

Figure 4: I²C Scanner

After the addresses are returned, the devices can now be addressed individually. In order to use I^{2}C with the LCDs, the correct library must be included:

```
#include <LiquidCrystal_I2C.h>
```

After including the library, the final step is to initialize the LCDs. Although there are now only two wires going to each LCD, the I^{2}C module still needs to know which pins on the LCD require what data. The first part names each LCD so they can be more easily addressed later. After this, the address of each LCD is entered as well as the pins the I^{2}C module needs to send information to. The final part is telling the I^{2}C module to turn the LCD backlight on.

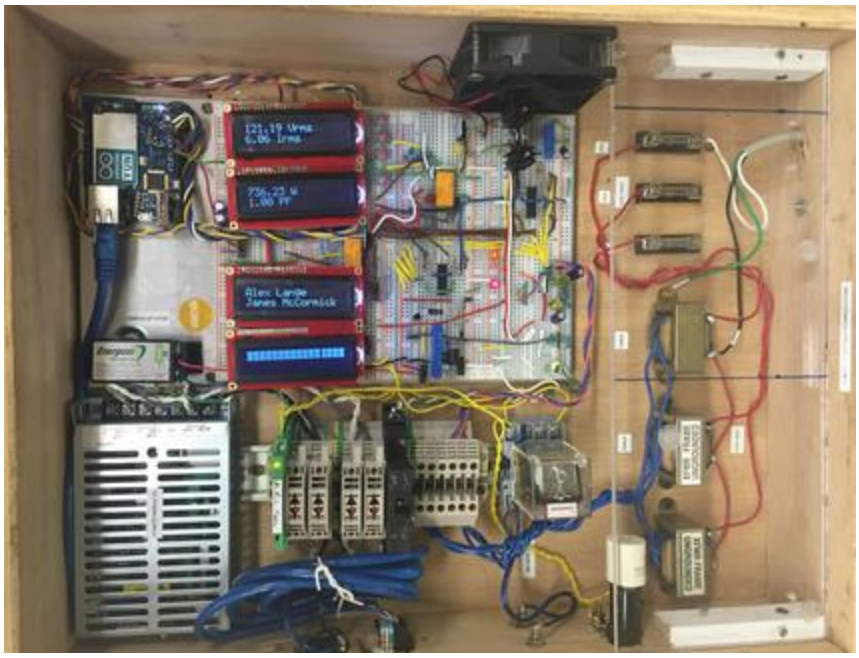
```
LiquidCrystal_I2C lcd1(0x25, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
LiquidCrystal_I2C lcd2(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
LiquidCrystal_I2C lcd3(0x23, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
LiquidCrystal_I2C lcd4(0x26, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
lcd1.begin(16,2);
lcd2.begin(16,2);
lcd3.begin(16,2);
lcd4.begin(16,2);
```

The LCDs can now easily be written to using the following command, where ‘lcd2’ can be changed to any of the assigned LCD names:

```
lcd2.print("I2C Works!! ");
```

Conclusion:

I^{2}C can be extremely useful when there is a desire to use a lot of different devices. It allows for future expansion as it saves on the number of I/O that are used. An example of its usefulness is using I^{2}C to address 4 different LCDs, Figure 5, to display 3-phase power information, the remaining I/O are then used for sampling buttons as well as exterior controls, like turning on a capacitor bank if the power factor drops too low.


121.19 Vrms
6.06 Irms
736.23 W
1.00 PF
Alex Lambe
James McCormick

References:

https://www.arduino.cc/en/Tutorial/HelloWorld

https://en.wikipedia.org/wiki/I%C2%B2C

http://www.egr.msu.edu/classes/ece480/capstone/fall15/group03/index.html