UG108: Praxis II
Asian Institute of Technology
Handout: I2C interface

January 2014
Undergraduate Program
Instructor: Waqar S. Qureshi, Matthew N. Dailey

I2C interface Tutorial

Introduction:

In this tutorial we will learn how to use I2C interface by using Arduino Uno Rev3 with *wire.h* library. I2C stands for ”Inter-Integrated Circuit”, which was invented and developed by Philips’ semiconductor division (now NXP) in the late 1970s. They need to reduce the number of data lines that travel between various integrated circuits. I2C interface is solution because it consists of only two lines where called **SDA** is carrying the data bits and the second called **SCL** is used as a clock signal. Figure 1 shows I2C bus wiring.

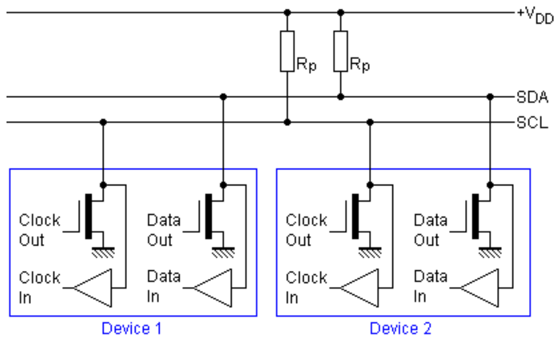


Figure 1: I2C bus wiring, reproduced from `http://www.lammertbies.nl/comm/info/I2C-bus.html`

If you are only using one I2C device, the pull-up resistors are (normally) not required. However, if you are running several devices, use two 10 kilo ohm resistors. The maximum length of an I2C bus is around one meter.

As there are several components that use the I2C interface! and our Arduino boards can control them all. There are many applications, such a real-time clocks, digital potentiometers, temperature sensors, digital compasses, memory chips, FM radio circuits, I/O expanders, LCD controllers, amplifiers, and so on. And you can have more than one on the bus at any time, in fact the maximum number of I2C devices used at any one time is 112.

Each device can be connected to the bus in any order, and devices can be masters or slaves. In our Arduino situation, the board is the master and the devices on the I2C bus are the slaves. We can write data to a device, or read data from a device. Each device has a unique address (7-bit address). We use that address in the functions to direct our read or write requests to the correct device. It is possible to use several devices with identical addresses on an I2C bus.

I2C Communication Protocol I2C has a master/slave protocol. The master initiates the communication. The sequences of events are:

1. The Master device issues a START condition. This condition informs all the slave devices to listen on the serial data line for instructions.
2. The Master device sends the 7-bit address of the target slave device and a read/write flag.
3. The Slave device with the matching address responds with an ACK signal.
4. Communcation proceeds between the Master and the Slave on the data bus. Both the master and slave can receive or transmit data depending on whether the communcation is a read or write. The transmitter sends 8-bits of data to the receiver which replies with a 1-bit ACK

5. When the communication is complete, the master issues a STOP condition indicating that everything is done.

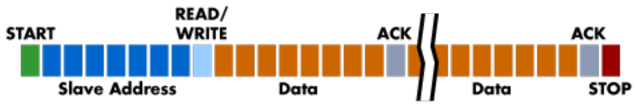


Figure 2: I2C communication protocal, reproduced from `http://www.totalphase.com/support/kb/10037/#theory`

I2C interface with Arduino

From a hardware perspective, the wiring is very easy. On the Arduino boards with the R3 layout (1.0 pinout), the SDA (data line) and SCL (clock line) are on the pin headers close to the AREF pin. We can use wire.h library that allow communicating with I2C / TWI devices. Then use the function *Wire.begin();* inside of void *setup()*.

Sending data from our Arduino to the I2C devices requires two things: the unique device address (we need this in hexadecimal) and at least one byte of data to send. For example, the address of the part in below is 00101111 (binary), which is 0x2F in hexadecimal. Then we want to set the wiper value, which is a value between 0 and 127, or 000 and 0x7F in hexadecimal. So to set the wiper to zero, we would use the following three functions:

```
Wire.beginTransmission(0x2F);
```

This sends the device address down the SDA (data) line of the bus. It travels along the bus, and "notifies" the matching device that it has some data coming

```
Wire.write(69); // sends 69 down the bus
```

This sends the byte of data to the device - into the device *register* (or memory of sorts), which is waiting for it with open arms. Any other devices on the bus will ignore this. Note that you can only perform one I2C operation at a time! Then when we have finished sending data to the device, we "end transmission". This tells the device that we're finished, and frees up the I2C bus for the next operation:

```
Wire.endTransmission();
```

Receiving data from an I2C device into our Arduino requires two things: the unique device address (we need this in hexadecimal) and the number of bytes of data to accept from the device. Receiving data at this point is a two stage process.

We now need to ask the device for the data, and how many bytes we want. For example, if a device held three bytes of data, we would ask for three, and store each byte in its own variable (for example, we have three variables of type byte: a, b, and c. The first function to execute is:

```
Wire.requestFrom(device_address, 3);
```

which tells the device to send three bytes of data back to the Arduino. We then immediately follow this with:

```
a = Wire.read();
b = Wire.read();
c = Wire.read();
```

We do not need to use *Wire.endTransmission()* when reading data. Now that the requested data is in their respective variables, you can treat them like any ordinary byte variable.

Experiment with PCF8574, expander 8-bit I/O

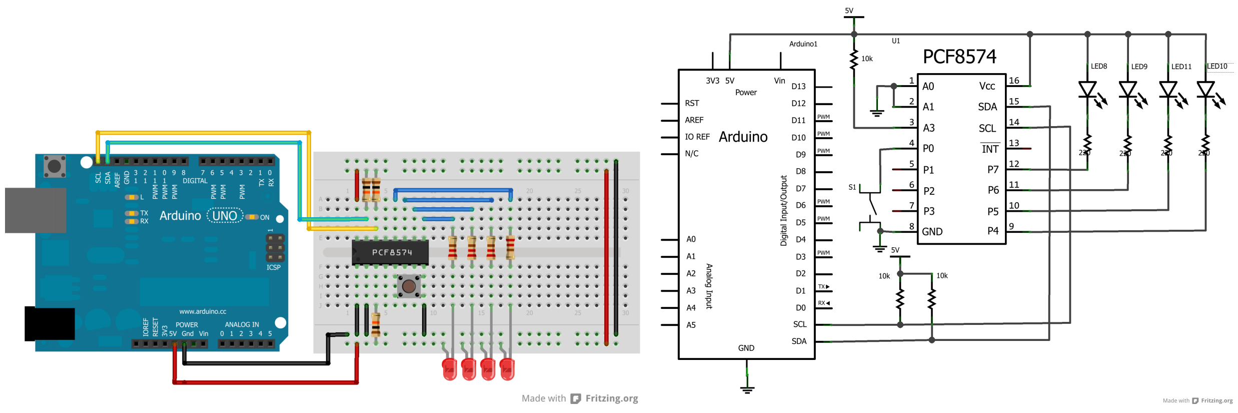
PCF8574: is IC that provides general purpose remote I/O expansion for most microcontroller families via the two-line bidirectional bus I2C. The device consists of an 8-bit quasi-bidirectional port and an I2C-bus interface. The PCF8574 has a low current consumption and includes latched outputs with high current drive capability for directly driving LEDs. It also possesses an interrupt line (INT) which can be connected to the interrupt logic of the microcontroller. By sending an interrupt signal on this line, the remote I/O can inform the microcontroller if there is incoming data on its ports without having to communicate via the I2C-bus. This means that the PCF8574 can remain a simple slave device. The PCF8574 and PCF8574A versions differ only in their slave address.

Components required

1. Arduino UNO R3
2. Proto-board
3. Jumper (Connecting wires)
4. IC PCF8574A
5. Resistor 220, 10kohm.
6. Button switch
7. LED

Reading Input of PCF8574A and display RS232

You can connect the circuit as in the Figure 3 where as the button switch is connected to P0-pin of PCF8574.



(a) Diagram generated by Fritzing-0.7.11 (b) Schematics generated by Fritzing-0.7.11

Figure 3: Expand I/O port with Arduino

```
#include<Wire.h>
#define PCF8574 B00100000       //PCF8574 ID code
#define PCF8574A B00111000  //PCF8574A ID code
//input
byte inpRead;
byte inpBuff = 0x0F;     //Don't care LED port
void setup(){
  Wire.begin();
  Serial.begin(9600);  //Start serial for output
}
void loop(){
  inpRead = ReadInput(PCF8574A);
  inpBuff = inpRead&0x01;       //Check P0 pin
  Serial.println(inpBuff,HEX);
  delay(500);
}
byte ReadInput(int IDCode)
{
  Wire.requestFrom(IDCode,1);  //1 byte Read
  return(Wire.read());
}
```

Result and Conclusion: This example shows the result of reading input of PCF8574 through RS232. The program displays the input data from Input Port of PCF8574 in HEX format. The value is changed when button is pressed.

Control output LEDs using PCF8574A

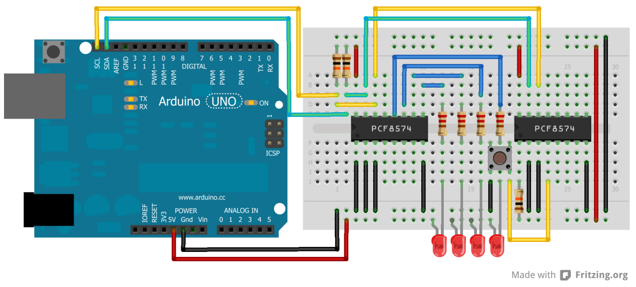
You can connect the circuit as in the Figure 3 where as four LEDs are connected to P4-P7 pins.

```
#include<Wire.h>
#define PCF8574 B00100000     //PCF8574 ID code
#define PCF8574A B00111000    //PCF8574A ID code
//LED control
int out[4] = {0x10, 0x20, 0x40, 0x80};
void setup(){
  Wire.begin();
}
void loop(){
//Run LED
for(int i=0; i<4;i++)
{
 // WriteOutput(PCF8574A, out[i]|0x0F);
  WriteOutput(PCF8574A, ~out[i]|0x0F);  //drive output without effect with input port
  delay(250);
}
}
void WriteOutput(int IDCode, byte output)
{
  Wire.beginTransmission(IDCode); //Send ID Code
  Wire.write(output);             //Send Data
  Wire.endTransmission();         //Stop Condition
}
```

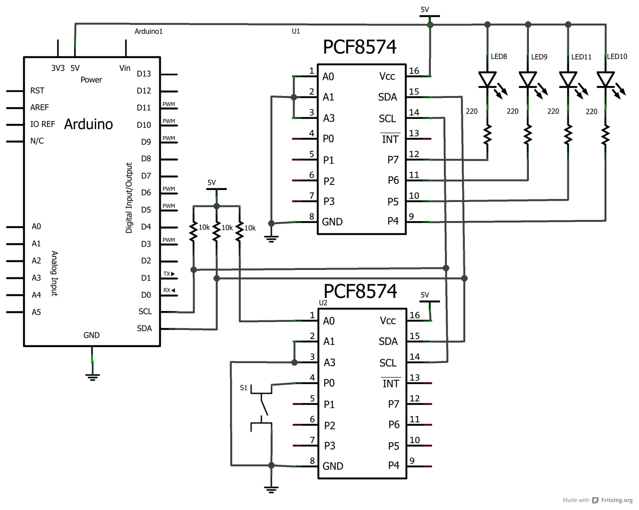
Result and Conclusion: After upload program to Arduino board, 4-LEDs is running which each steps delays for 250 ms. If either SDA or SCL line is cut off, the running LED is stopped because there is no transmissed data to PCF8574.

Assignment

Write Arduino program to run four LEDs by using the first PCF8574-IC which its address {A2,A1,A0} is {0,0,0}. When the button is pressed, all LEDs are blinked in every 1 second. The button is connected at P0 pins of the second PCF8574-IC which its address {A2,A1,A0} is {0,0,1}. The given code is not complete, you need to finish.



(a) Diagram generated by Fritzing-0.7.11



(b) Schematics generated by Fritzing-0.7.11

Figure 4: Expand I/O port with Arduino

```
//Experiment of PCF8574 expander I/O by using I2C interface
//Connect two PCF8574 ICs with different addresses

#include<Wire.h>
#define PCF8574   B00100000   //PCF8574 ID code
#define PCF8574A  B00111000   //1st-PCF8574A ID code (A0=0, A1=0, A2=0);
#define PCF8574A2 B00111001   //2nd-PCF8574A ID code (A0=1, A1=0, A2=0);

int out[4] = {0x10, 0x20, 0x40, 0x80};
int out2 = 0xF0;

byte inpRead;
byte inpBuff = 0x0F;    //Don't care last 4-bit

void setup(){
  Wire.begin();
}

void loop(){
  inpRead = ReadInput(PCF8574A2);  //read input from 2nd-PCF8574
  inpBuff = inpRead&0x01;          //check P0 Pin
  if(inpBuff==0){                  //button is pressed
    WriteOutput(PCF8574A, (~out2)|0x0F);
    out2 = ~out2;
    delay(1000);
  }
  else{
    //Run LED
    for(int i=0; i<4;i++)
    {
      WriteOutput(PCF8574A, ~out[i]|0x0F);
      delay(250);
    }
  }
}
```

Result and Conclusion:

References

- http://www.lammertbies.nl/comm/info/I2C-bus.html
- http://startingelectronics.com/beginners/start-electronics-now/tut18-two-wire-arduino-knight-rider
- http://blog.littlebirdelectronics.com/tutorial-arduino-and-the-i2c-bus
- http://hobbybotics.com/projects/hobbybotics-pcf8574a-i2c-io-expander
- http://www.totalphase.com/support/kb/10037/#theory