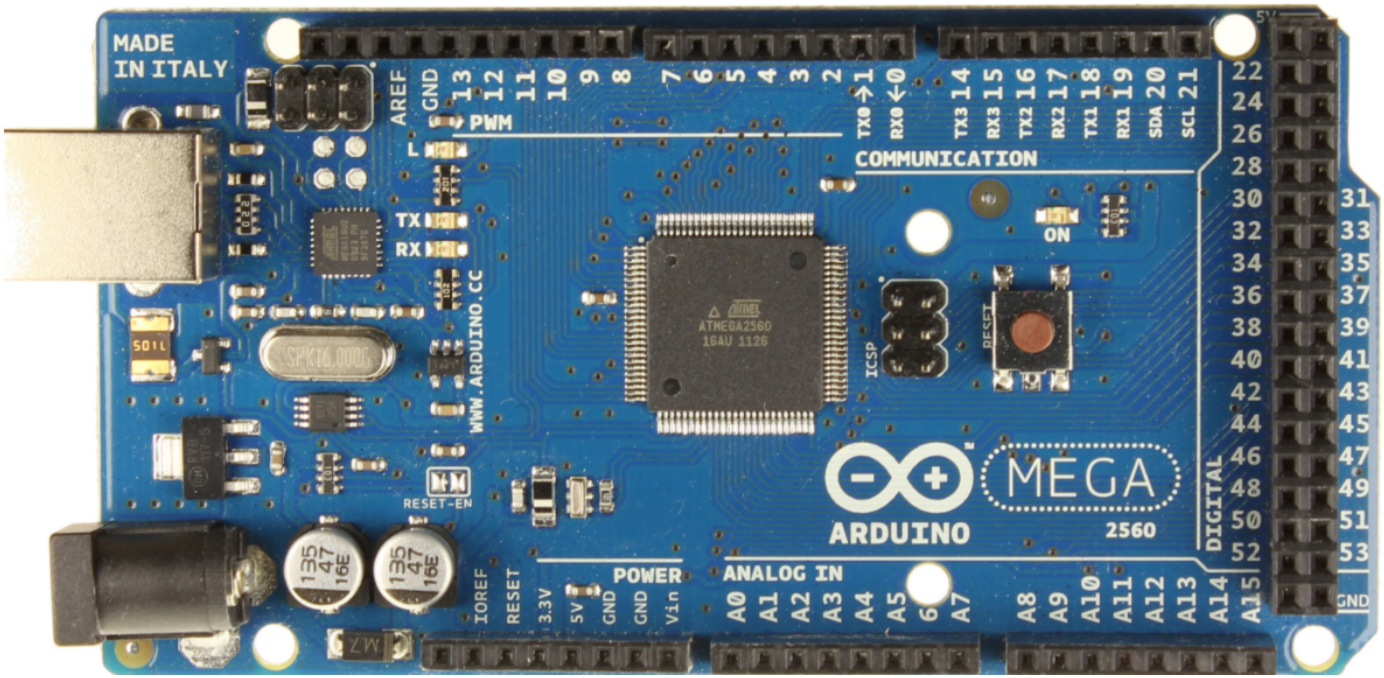
Slide Index

Week 4 – 102 slides Michel Hanbury Sept. 18th 2018

Fall 2018 CAM8302E (Week 4) Microcomputer Interfacing

Basic I/O Hardware for the Arduino



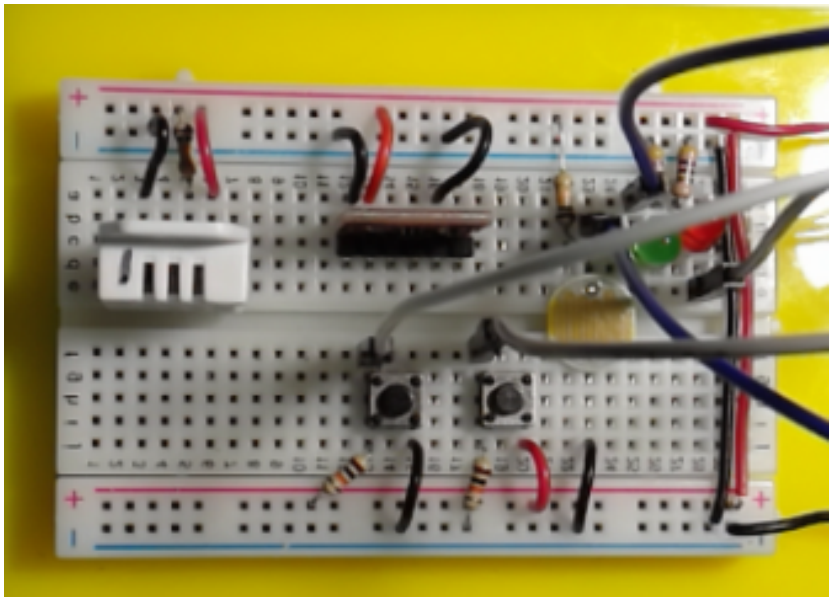
Basic_In_Out_Ain_2pB_F2016

```
1  /* Michel Hanbury September 28th 2016
2   Basic IO test  2 PB, 2 LEDs, and Photocell  */
3
4  int GrnLed = 22;      // definitions
5  int RedLed = 24;
6  int BoardLed = 13;
7  int Pb1 = 26;
8  int Pb2 = 28;
```

The first few lines of the Arduino program define labels used later in the program.

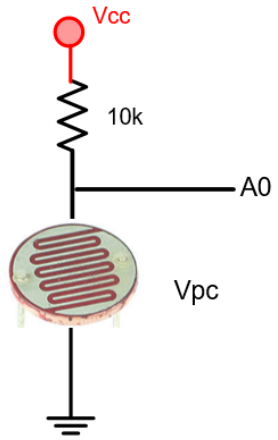
For example if the label “GrnLed” is used it is actually replaced with the value 22.

```
10 void setup() {
11
12   Serial.begin(9600);   // configure serial I/O
13
14   pinMode(GrnLed, OUTPUT);
15   pinMode(RedLed, OUTPUT);
16   pinMode(BoardLed, OUTPUT);
17
18   pinMode(Pb1, INPUT);
19   pinMode(Pb2, INPUT);
20 }
21
```



Configure Input and Output pins. To protect the I/O hardware and the Arduino the I/O pins are defaulted as inputs until the software configures them as outputs.

```
22 void loop() {
23    float Vsup = 4.61;  // measured with meter
24    int Photocell = analogRead(A8);
25    int Pb_state1 = digitalRead(Pb1);
26    int Pb_state2 = digitalRead(Pb2);
27
28    float Vpc = Photocell * (Vsup / 1023.0);  // Vin to A/D
29
30    float I = (Vsup - Vpc)/10000;
31
32    float Rpc = Vpc / I;
33
34    float ImA = 1000 * (Vsup - Vpc)/10000;
```



Calculations on data from photocell. The code converts the integer value from the A/D converter to a voltage and then to a resistance.

```
35
36    if (Pb_state1 == LOW)
37      digitalWrite(GrnLed,HIGH);
38    else
39      digitalWrite(GrnLed,LOW);
40
41    if (Pb_state2 == HIGH)
42      digitalWrite(RedLed,HIGH);
43    else
44      digitalWrite(RedLed,LOW);
45
46    if (Rpc  > 2000.0)    //  Getting dark
47      digitalWrite(BoardLed,HIGH);
48    else
49      digitalWrite(BoardLed,LOW);
50
```

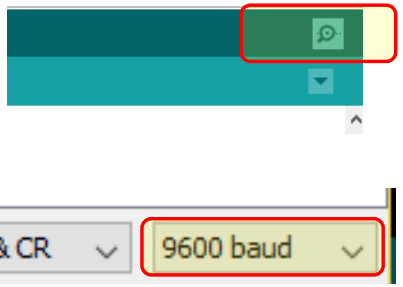
PB1 push button is normally HIGH using a pull up resistor. When the push button is pressed the pins short and a connection to ground is made.

Control output based on input conditions.

PB2 is normally LOW, a resistor is used to pull the signal to a low and a short on the pin ties it to +5.

Pin 13 L on the board.

```
51   Serial.print("  PhotoCell Integer =    ");
52   Serial.print(Photocell);
53
54   Serial.print("  PhotoCell Volts =    ");
55   Serial.print(Vpc,4);
56
57   Serial.print("  PhotoCell Current mA =    ");
58   Serial.print(ImA,4);
59
60   Serial.print("  PhotoCell Resistance Ohms =    ");
61   Serial.print(Rpc);
62
63   Serial.print("   Pb1 state =    ");
64   Serial.print(Pb_state1);
65
66   Serial.print("   Pb2 state =    ");
67   Serial.println(Pb_state2);
68   delay (50);
69 }
```



The Arduino Serial monitor can be used to output variable data to a computer screen.

The BPS (bits/second) baud rate must match the configuration in software. In this example the rate is 9600 bits per second.

Serial.println moves to the start of the next line, then prints.

```
PhotoCell Integer =  79  PhotoCell Volts =  0.3560  PhotoCell Current mA =  0.4254  PhotoCell Resistance Ohms =  836.86  Pb1 state =  1  Pb2 state =  0
PhotoCell Integer =  79  PhotoCell Volts =  0.3560  PhotoCell Current mA =  0.4254  PhotoCell Resistance Ohms =  836.86  Pb1 state =  1  Pb2 state =  0
PhotoCell Integer =  79  PhotoCell Volts =  0.3560  PhotoCell Current mA =  0.4254  PhotoCell Resistance Ohms =  836.86  Pb1 state =  1  Pb2 state =  0
PhotoCell Integer =  79  PhotoCell Volts =  0.3560  PhotoCell Current mA =  0.4254  PhotoCell Resistance Ohms =  836.86  Pb1 state =  1  Pb2 state =  0
PhotoCell Integer =  79  PhotoCell Volts =  0.3560  PhotoCell Current mA =  0.4254  PhotoCell Resistance Ohms =  836.86  Pb1 state =  1  Pb2 state =  0
PhotoCell Integer =  79  PhotoCell Volts =  0.3560  PhotoCell Current mA =  0.4254  PhotoCell Resistance Ohms =  836.86  Pb1 state =  1  Pb2 state =  0
PhotoCell Integer =  79  PhotoCell Volts =  0.3560  PhotoCell Current mA =  0.4254  PhotoCell Resistance Ohms =  836.86  Pb1 state =  1  Pb2 state =  0
```

```
float Vpc = Photocell * (Vsup / 1023.0);  // Vin to A/D

float I = (Vsup - Vpc)/10000;

float Rpc = Vpc / I;

float ImA = 1000 * (Vsup - Vpc)/10000;
```

Check the results based on the code.

```
PhotoCell Integer =  319  PhotoCell Volts =  1.4375  PhotoCell Current mA =  0.3172  PhotoCell Resistance Ohms =  4531.25  Pb1 state =  1  Pb2 state =  0
PhotoCell Integer =  316  PhotoCell Volts =  1.4240  PhotoCell Current mA =  0.3186  PhotoCell Resistance Ohms =  4469.59  Pb1 state =  1  Pb2 state =  0
PhotoCell Integer =  312  PhotoCell Volts =  1.4060  PhotoCell Current mA =  0.3204  PhotoCell Resistance Ohms =  4388.19  Pb1 state =  1  Pb2 state =  0
PhotoCell Integer =  312  PhotoCell Volts =  1.4060  PhotoCell Current mA =  0.3204  PhotoCell Resistance Ohms =  4388.19  Pb1 state =  1  Pb2 state =  0
PhotoCell Integer =  314  PhotoCell Volts =  1.4150  PhotoCell Current mA =  0.3195  PhotoCell Resistance Ohms =  4428.77  Pb1 state =  1  Pb2 state =  0
PhotoCell Integer =  314  PhotoCell Volts =  1.4150  PhotoCell Current mA =  0.3195  PhotoCell Resistance Ohms =  4428.77  Pb1 state =  1  Pb2 state =  0
```

```
28    float Vpc = Photocell * (Vsup / 1023.0);  // Vin to A/D
29
30    float I = (Vsup - Vpc)/10000;
31
32    float Rpc = Vpc / I;
33
34    float ImA = 1000 * (Vsup - Vpc)/10000;
```

Check the results based on the code.

LCD Display and 5 key Analog Keypad

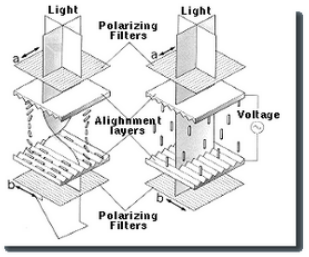
Liquid Crystal Display (LCD)

LCD – Liquid Crystal Display

- Low Power (mA)
- Easy to use libraries available
- Inexpensive and common.
- Easy to wire (8 pins)

A combination of polarizing filters and twisted liquid crystal creates a liquid crystal display.

When two polarizing filters are arranged along perpendicular polarizing axes, light entering from above is re- directed 90 degrees along the helix arrangement of the liquid crystal molecules so that it passes through the lower filter.



When voltage is applied, the liquid crystal molecules straighten out of their helix pattern and stop redirecting the angle of the light, thereby preventing light from passing through the lower filter.

LCD Display 16 x 2

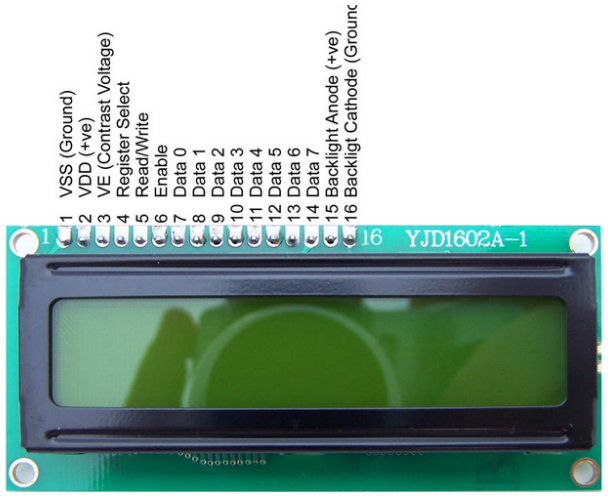
I2C serial
LCD Display
20 char x 4 lines

A special circuit attached to the rear of the display converts the display from parallel to I2C serial. Connecting to this display requires only 4 wires.

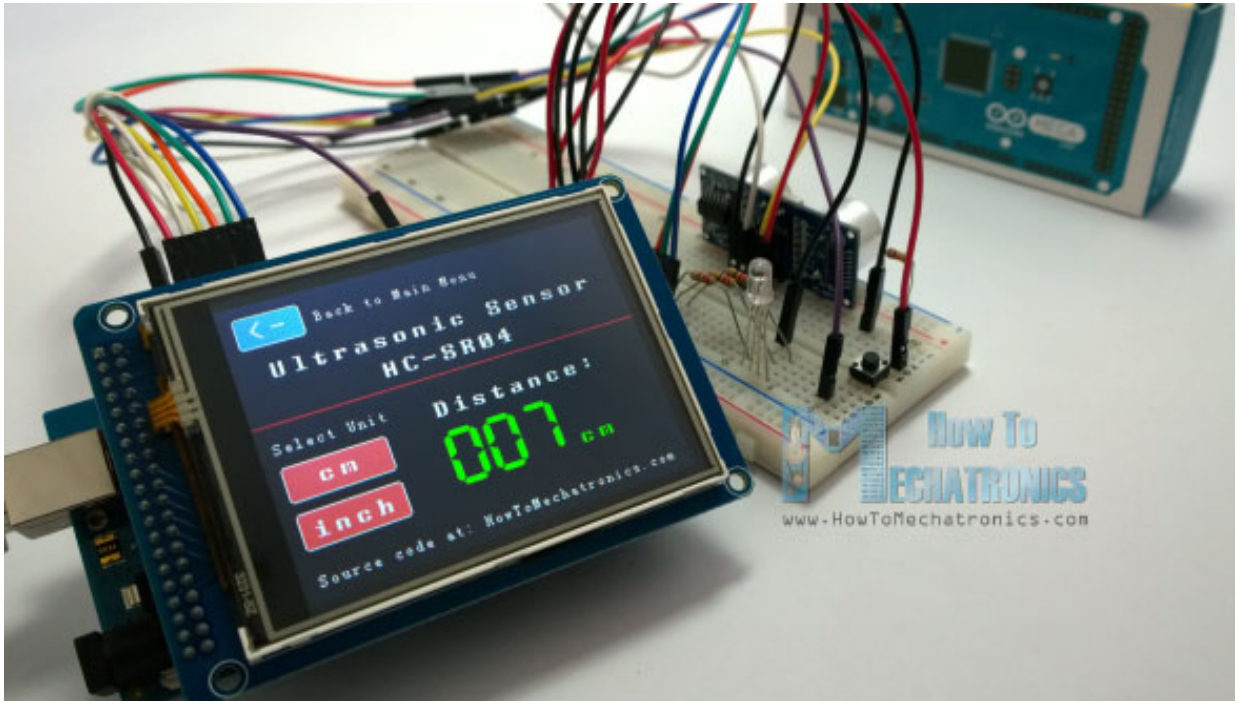
The display includes a backlight and a contrast adjust.

LCD Display
20 char x 4 lines
Parallel Interface.

This is the type that comes with the lab kit. It is less expensive and the software is less complex than the I2C type



Touch colour display. Adds input and output to an embedded system. A great way to create a basic HMI. You can add sliders, gauges and graphics. Programming is more complicated. Most displays have drivers and functions that make programing easier.



Parallel LCD Display 16 char. By 2 lines.

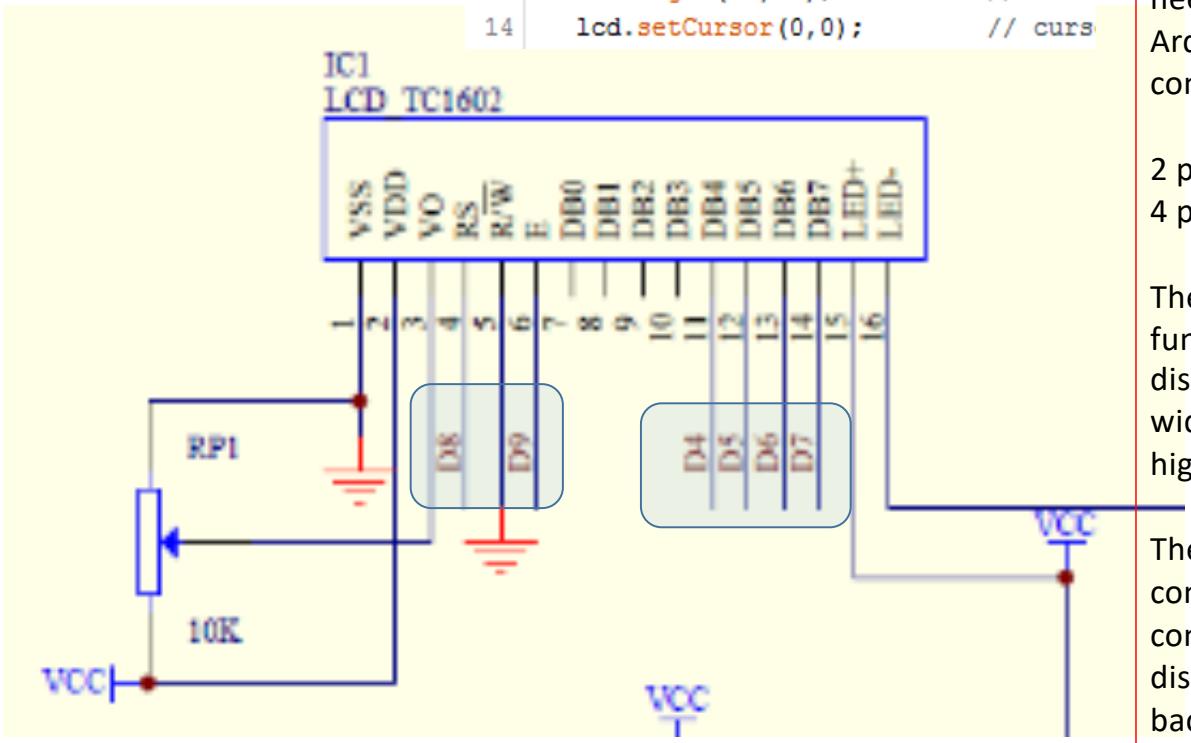
```
10 LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
11
12 void setup(){
13    lcd.begin(16, 2);          // star
14    lcd.setCursor(0,0);        // curs
```

This LCD display uses a 4 bit parallel interface. The RS and E are signals used to control data.

Only the upper 4 bits of the display are used by the Arduino.

The “E” pin is the clock signal.

The “RS” is the register select and selects between data and control.

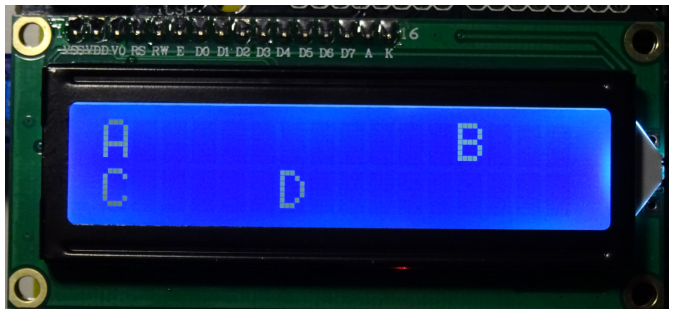


The LCD function needs to know which Arduino pins will control the LCD.

2 pins for control and 4 pins for data.

The (16,2) tells the function that the display is 16 character wide and 2 characters high.

The resistor is used to control the display contrast. Some displays include a backlight,



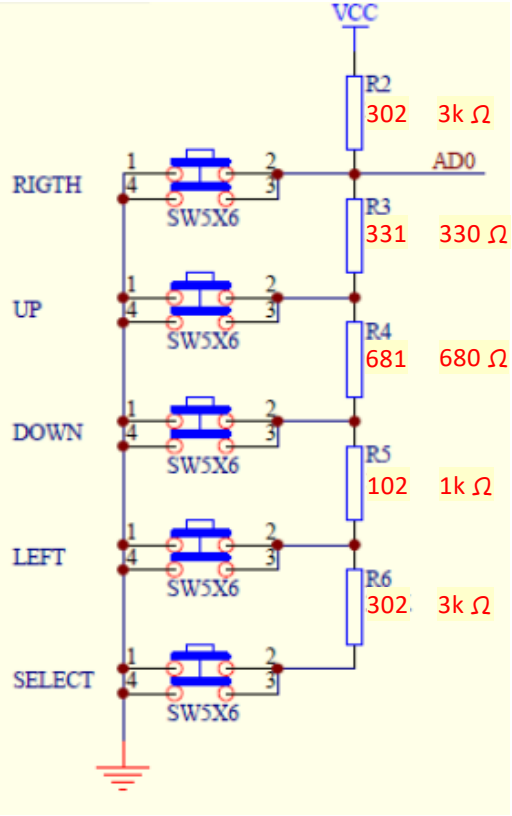

```
19 void loop(){
20
21     lcd.setCursor(0,0);
22     lcd.print("A");
23     lcd.setCursor(12,0);
24     lcd.print("B");
25     lcd.setCursor(0,1);
26     lcd.print("C");
27     lcd.setCursor(6,1);
28     lcd.print("D");
29
```

The LCD display is arranged as 2 rows and 16 columns. The numbering starts at 0 for the rows and columns. The first number represents the column the second the row.
The top left character is at 0,0.
For example 6,1 is the bottom row and the 7th position over.

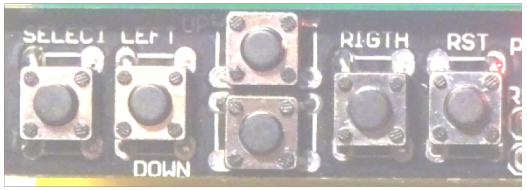
Arduino LCD Setup()

```
8  #include <LiquidCrystal.h>
9
10 LiquidCrystal lcd(8, 9, 4, 5, 6, 7);          // select the pins used on the LCD panel
11
12 void setup(){
13    lcd.begin(16, 2);          // start the library   2 lines 16 characters
14    lcd.setCursor(0,0);        // cursor to top lef
15    lcd.print("UP or DN");     // display message
16    delay(4000);               // wait 4 seconds
17    lcd.clear();               // clear lcd display
18    lcd.setCursor(0,0);        // set the LCD cursor position top line left
19
20   }
```

The Arduino IDE includes a library for an LCD display with an HD44780 controller. LiquidCrystal lcd(8, 9, 4, 5, 6, 7) defines the pins that are required by the display. The display uses 4 data lines, an enable signal and a RS - register select pin that selects between either data or control signals. The display uses a Hitachi HD44780 controller, this is the most common type. You must include the “LiquidCrystal.h” header file in your Arduino sketch to use the LCD functions.



Arduino LCD / Shield Keypad



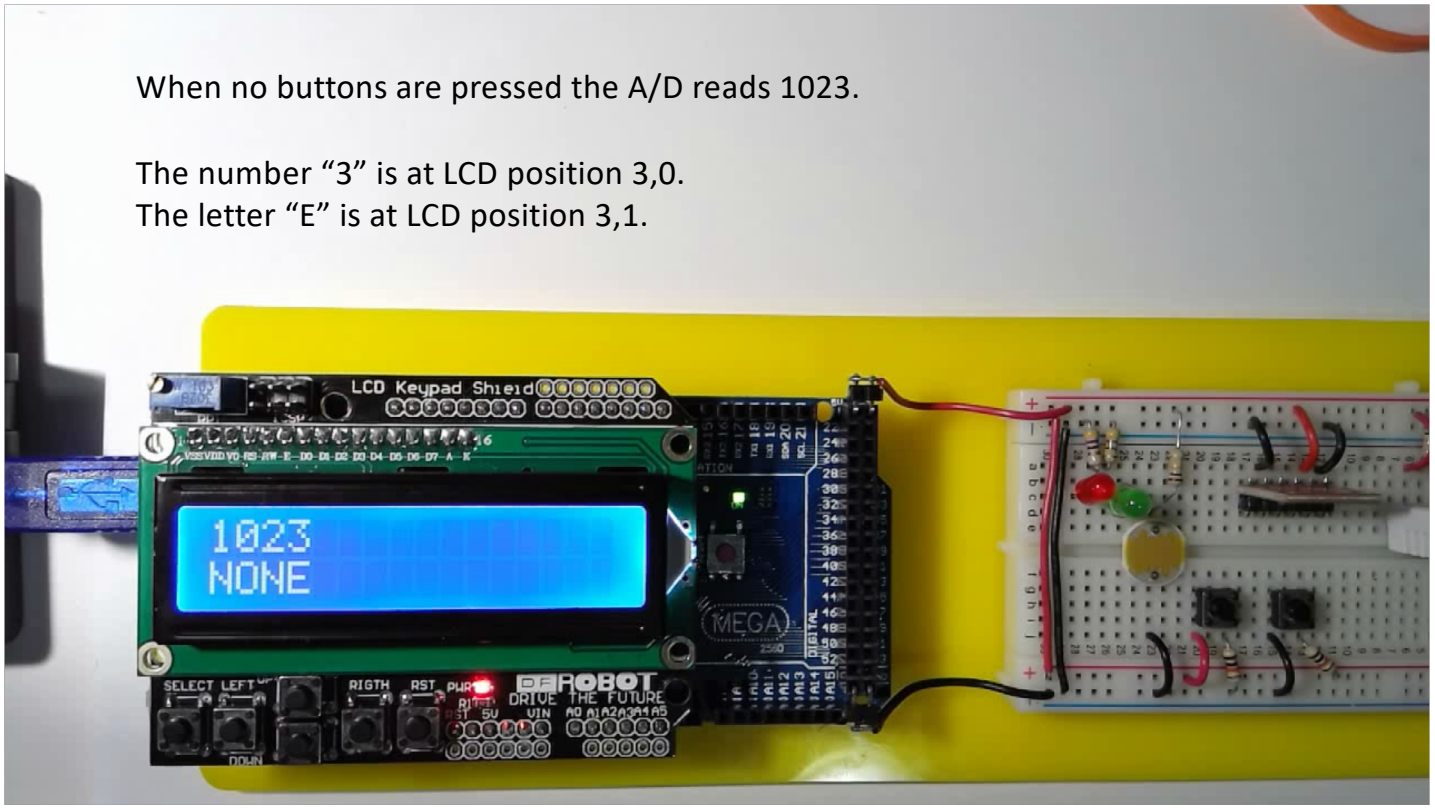
The LCD display board also has a set of push button switches which are connected to a set of resistors used in a voltage divider. Pressing different switches will cause the voltage on Analog 0 (AD0) to change. The resistors shown are the actual values used on the board.

The numbers in red are values I found when testing my keypad. The range may vary by +/- 3.

Right – 0
Up – 99
Down – 257
Left – 411
Select 641

When no buttons are pressed the A/D reads 1023.

The number “3” is at LCD position 3,0.
The letter “E” is at LCD position 3,1.



Arduino LCD / Keypad Shield

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);          // select the pins used on the LCD panel

void setup(){
   lcd.begin(16, 2);          // start the library 2 lines 16 characters

   lcd.clear();               // clear lcd display
   lcd.print("CAM8302E");

  }
void loop(){

      int sensorValue = analogRead(0);  read value from divider

      lcd.setCursor(0,1);             // set the LCD cursor position lower line left
      lcd.print(sensorValue);
      lcd.print("     ");

      delay(250);               // wait half a second
    }
```

Read and display the value from analog input 0.

Example – Setup for LCD and I/O

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);    // select the pins used on the LCD panel
#define PB 22
#define LED 24

void setup(){

   pinMode(PB,INPUT);
   pinMode(LED,OUTPUT);

   lcd.begin(16, 2);          // start the library 2 lines 16 characters

   lcd.clear();               // clear lcd display
   lcd.print("Display PB Value");
  }
```

Using the push-buttons to make decisions

```
22 void loop(){
23     float supply = 5.0;
24     float sensorValue = analogRead(0);
25     float sensorVoltage = sensorValue/1023 * supply;
26
27      if ((sensorValue>89)&(sensorValue<109))   //check for UP switch
28  {
29       lcd.setCursor(0,1);             // set the LCD cursor position lower line left
30       lcd.print(sensorValue);
31       lcd.print("    ");
32  }
33
34     if   ((sensorValue>247)&(sensorValue<267)) // check for DN switch
35 {
36       lcd.setCursor(0,1);             // set the LCD cursor position lower line left
37       lcd.print(sensorVoltage);
38       lcd.print("     ");
39  }
40
41     delay(250);              // wait half a second
42     }
```

Right – 0
Up – 99
Down – 257
Left – 411
Select 641

This program reads the analog 0 input voltage to determine if the sensor voltage.

The program uses a range to determine which key is being pressed.

A range is chosen because the resistor have a tolerance of +/- 5%

Example – Using the LCD display to show the value of analog Input 8.

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);        // select the pins used on the LCD panel

void setup(){
   lcd.begin(16, 2);          // start the library 2 lines 16 characters

   lcd.clear();               // clear lcd display
   lcd.print("Read A8");

  }
void loop(){

    int sensorValue = analogRead(8); read value from divider

      lcd.setCursor(0,1);             // set the LCD cursor position lower line left
      lcd.print(sensorValue);
      lcd.print("     ");

      delay(250);                // wait half a second
    }
```

Read and display the value from analog input 8.

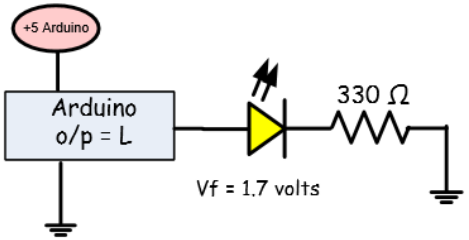
Example – Using the UP/DN buttons to control Outputs

```
24 void loop(){
25
26      boolean pbState = digitalRead(PB);
27       int sensorValue = analogRead(0);  // read value from keypad divider
28
29       lcd.setCursor(0,1);             // set the LCD cursor position lower line left
30       lcd.print(pbState);
31       lcd.print("     ");            // blank characters
32
33      if ((sensorValue>89)&(sensorValue<109))  //check for UP switch
34       digitalWrite(LED,HIGH);
35
36       if  ((sensorValue>247)&(sensorValue<267)) // check for DN switch
37       digitalWrite(LED,LOW);
38       delay(250);                 // wait a quarter second
39     }
40
```

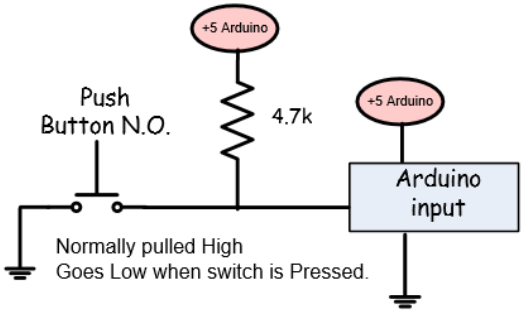
Example – Using the LCD display to show the value of analog Input 8.

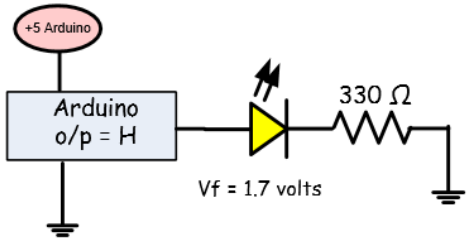
```
24 void loop(){
25
26      boolean pbState = digitalRead(PB);
27      int sensorValue = analogRead(0);  // read value from keypad divider
28
29      lcd.setCursor(0,1);             // set the LCD cursor position lower line left
30      lcd.print(pbState);
31      lcd.print("     ");            // blank characters
32
33
34      lcd.setCursor(8,1);             // set the LCD cursor position lower line left
35      lcd.print(sensorValue);
36      lcd.print("    ");            // blank characters
37
38
39      if ((sensorValue>89)&(sensorValue<109))  //check for UP switch 99 +/- 10
40      digitalWrite(LED,HIGH);
41
42      if  ((sensorValue>247)&(sensorValue<267)) // check for DN switch 257
43      digitalWrite(LED,LOW);
44      delay(250);               // wait a quarter second
45    }
```

Basic Arduino Inputs and Outputs
Transistor Switch



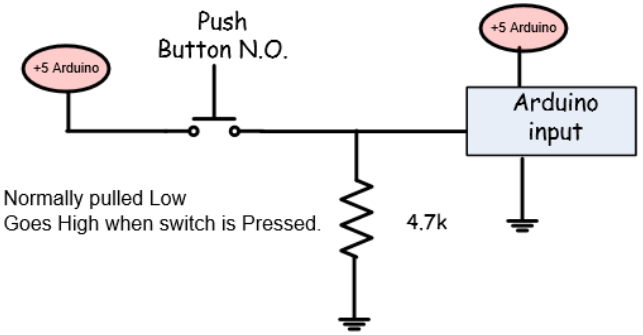
Led is reversed biased
Arduino output is about 0 volts.
0 volts across resistor.
No current flow.
Led Off.

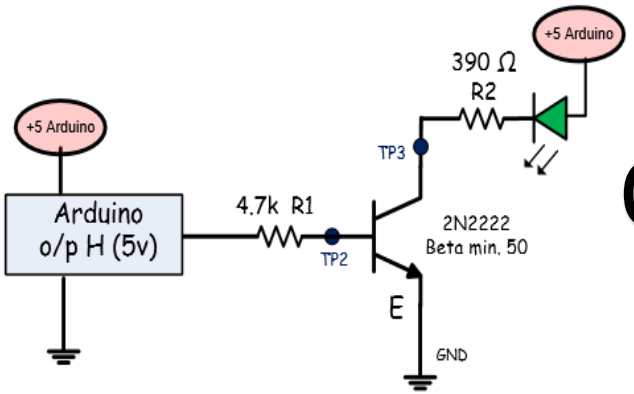




Led is forward biased
Arduino output is about 5 volts.
3.3 volts across resistor. I=V/R
3.3 volts / 330 ohms = 10 mA
Led On.

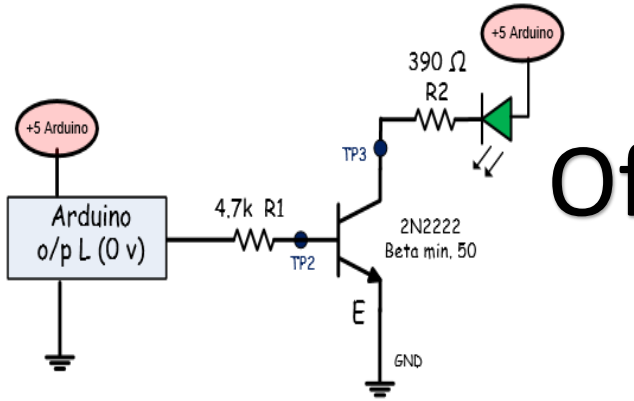
Pull up or Pull down prevent a floating condition.





On

1) The output from the Arduino is about 5.0 volts.
2) Current flows from the Arduino through the resistor, the base and the emitter diode.
3) The diode drop is 0.7 volts. The remaining 4.3 volts is across the 4.7k resistor.
4) 4.3v/4.7k = 0.91 mA of base current.
5) The transistor in on acting like a closed switch.
6) The collector voltage equals about 0.1 volts.
7) The LED is forward biased with a drop of about 1.7 volts.
8) 5.0 volts – 1.7 volts leaves about 3.3 volts across the 390 ohm resistor.
9) 3.3 volts/390 ohms = 8.5 mA.

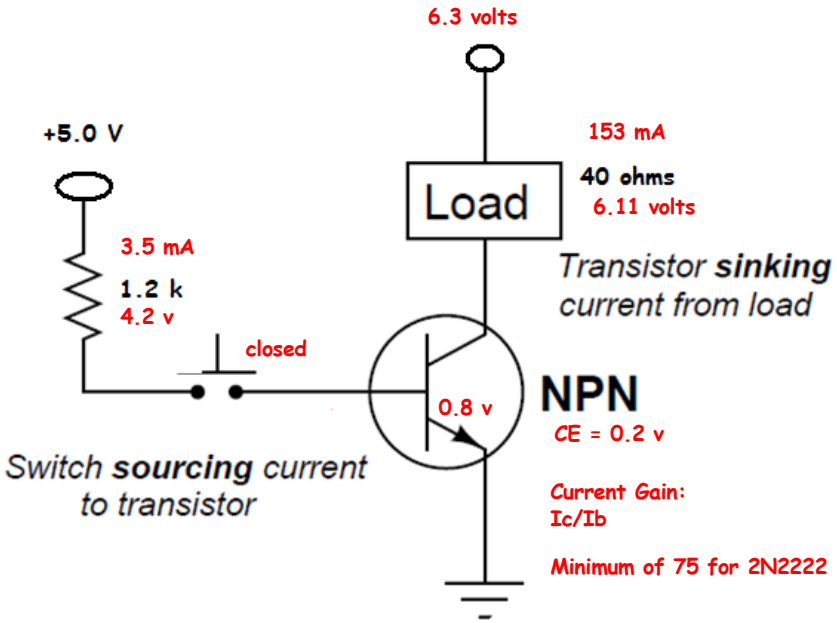


Off

1) The output from the Arduino is about 0 volts.
2) No current flows through the base.
3) The emitter diode is reversed biased.
4) 0 mA of base current.
5) The transistor in off acting like an open switch.
6) The collector voltage equals about 3.5 volt. (Transistor 3.5megs, LED 1.5 Megs of resistance.
7) The LED is reversed biased.
8) 0 volts across the 390 ohm resistor.
9) 0 volts/390 ohms = 0 mA.

NPN 2N2222 Transistor Switch – "ON"

- Saturation – Transistor is turned on: The collector to emitter junction acts like a closed switch.
- The base voltage must be greater than the emitter voltage by 0.8 volts.
- The transistor has a gain of about 75.
- Example: If the base current equals 3.5 mA and the gain = 75 mA the transistor will allow up to 263 mA of current to flow.
- Transistors can only handle a certain amount of collector current specified in the data sheets.
- The CE junction does not act like a perfect switch and may have a voltage across the CE junction of about 0.2 volts.

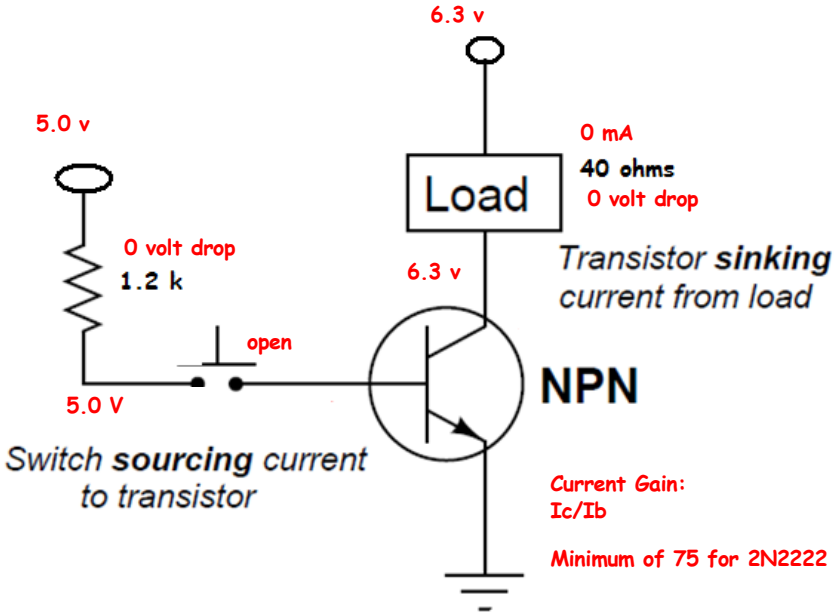


NPN 2N2222 Transistor Switch – "Off"

NPN Transistor Switch off:

- **Cut-off – Transistor is turned off: The collector to emitter junction acts like an open switch.**

- **The BE voltage must be less than 0.8 volts.**



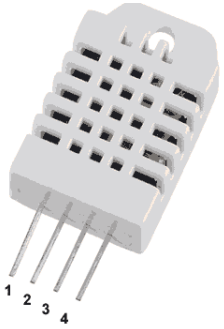
1 Wire Data -- Temperature and Humidity Sensor DHT22

DHT22 Digital Temperature and Humidity Sensor (Serial Data Output)

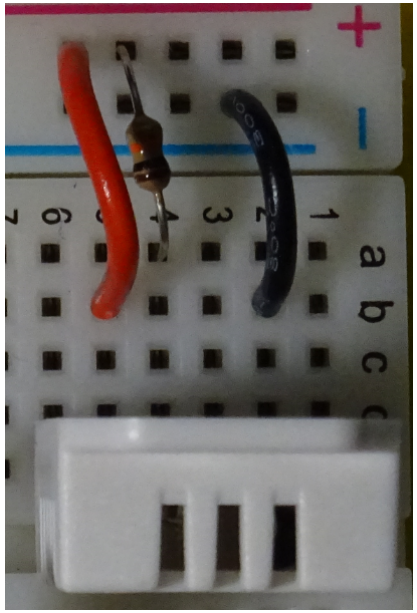
DESCRIPTION

The DHT22 is a basic, low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed). Its fairly simple to use, but requires careful timing to grab data. The only real downside of this sensor is you can only get new data from it once every 2 seconds, so when using our library, sensor readings can be up to 2 seconds old.

DHT22 pins	
1	VCC
2	DATA
3	NC
4	GND




```
21.50  deg C    42.80  % hum
21.50  deg C    42.70  % hum
21.50  deg C    42.60  % hum
21.50  deg C    42.50  % hum
21.40  deg C    42.40  % hum
21.40  deg C    42.40  % hum
21.40  deg C    42.40  % hum
```

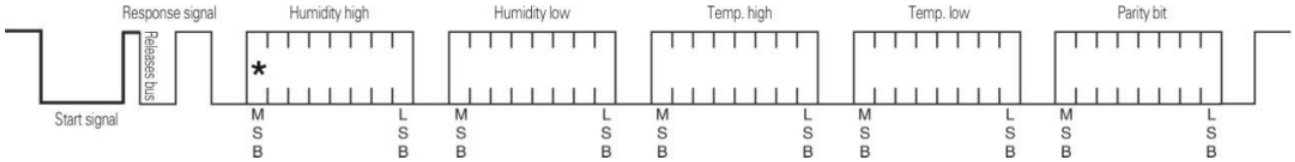


The temperature / humidity sensor connects the Arduino using a 1 wire serial interface. The sensor transmits 32 bits of data representing the % humidity and the temperature in degrees Celsius.

DHT22 Digital Temperature and Humidity Sensor (Serial Data Output)

◎Single bus to send data definition

SDA For communication and synchronization between the microprocessor and the AM2302, single–bus data format, a transmission of 40 data, the high first–out. Specific communication timing shown in Figure 5, the communication format is depicted in Table 5.



Pic5： AM2302 Single–bus communication protocol

The DHT22 temperature and humidity sensor sends a digital pattern to the Arduino representing % relative humidity and temperature in degrees Celsius .

The first 16 bits transmitted represent the humidity, the second 16 the temperature and the last 8 bits are a checksum to verify the data.

$$\text{Relative Humidity (RH)} = \frac{\text{(Actual Vapor Density)}}{\text{(Saturation Vapor Density)}} \times 100\%$$

relative humidity

Word Origin

noun

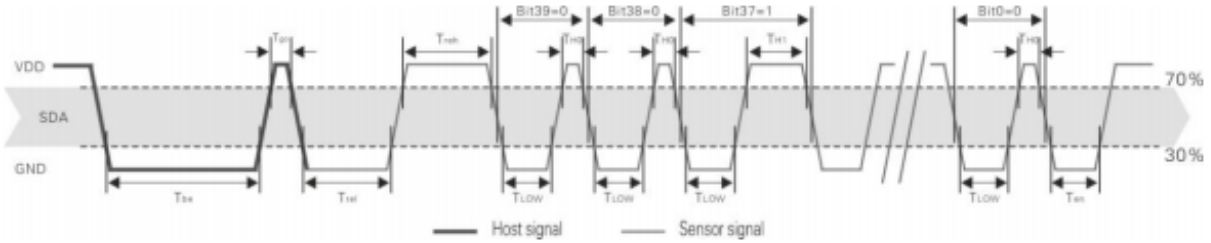
1. the amount of water vapor in the air, expressed as a percentage of the maximum amount that the air could hold at the given temperature; the ratio of the actual water vapor pressure to the saturation vapor pressure.
   Abbreviation: RH, rh.

AOSONG Temp、Humidity & Dew point measurement experts

7.3 Single-bus communication timing

User host (MCU) to send a start signal (data bus SDA line low for at least 800μ s) after AM2302 from Sleep mode conversion to high-speed mode.The host began to signal the end of the AM2302 send a response signal sent from the data bus SDA serial 40Bit's data, sends the byte high; data sent is followed by: Humidity high、Humidity low、Temperature high、Temperature low、Parity bit, Send data to the end of trigger information collection, the collection end of the sensor is automatically transferred to the sleep mode, the advent until the next communication.

Detailed timing signal characteristics in Table 6, Single-bus communication timing diagram Pic 6:



Pic 6: AM2302 Single-bus communication timing

Data from humidity and temperature sensor.

40 bit data

32 bits of data
8 bit checksum.

16 bits for humidity

16 bits temperature

DHT22 Example: Digital Temperature and Humidity Sensor (Serial Data Output)

◎Single-bus data calculation example

Example 1: 40 Data received:

0000 0010	1001 0010	0000 0001	0000 1101	1010 0010
High humidity 8	Low humidity 8	High temp. 8	Low temp. 8	Parity bit

Calculate:

0000 0010+1001 0010 +0000 0001+0000 1101= 1010 0010 (Parity bit)

Received data is correct:

humidity: 0000 0010 1001 0010 = 0292H (Hexadecimal)= $2 \times 256 + 9 \times 16 + 2 = 658$

=> Humidity = 65.8%RH

Temp.: 0000 0001 0000 1101 = 10DH(Hexadecimal) = $1 \times 256 + 0 \times 16 + 13 = 269$

=> Temp.= 26.9℃

◎Special Instructions:

When the temperature is below 0 ℃, the highest position of the temperature data.

Example: −10.1 ℃ Expressed as 1 000 0000 0110 0101

Temp.: 0000 0000 0110 0101 = 0065H(Hexadecimal)= $6 \times 16 + 5 = 101$

=> Temp. = −10.1℃

Example 2: 40 received data:

0000 0010	1001 0010	0000 0001	0000 1101	1011 0010
High humidity 8	Low humidity 8	High temp. 8	Low temp. 8	Parity bit

Calculate:

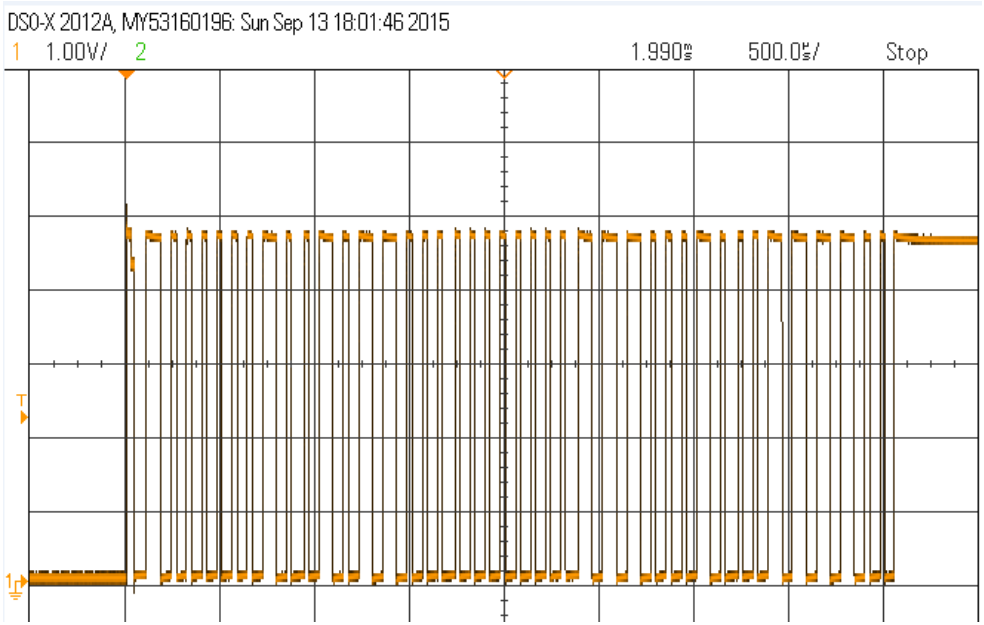
0000 0010+1001 0010 +0000 0001+0000 1101= 1010 0010 ≠ 1011 0010 (Validation error)

The received data is not correct, give up, to re-receive data.

DHT22 Example: Digital Temperature and Humidity Sensor (Serial Data Output)

```
28 void setup(){
29
30     lcd.begin(16, 2);               // start the library   2 lines 16 characters
31     dht.begin();
32  }
33
34 void loop(){
35    float h = dht.readHumidity();
36    float t = dht.readTemperature();
37
38     lcd.setCursor(0,0);             // set the LCD cursor   position top line left
39     lcd.print(t);
40     lcd.print("  deg C");
41
42     lcd.setCursor(0,1);       // move cursor to second line "1" and 9 spaces over
43     lcd.print(h);        // display value
44     lcd.print("  % humidity");
45
46     delay(500);               // wait half a second
47     }
```

DHT22 Example: Digital Temperature and Humidity Sensor (Serial Data Output)



This capture shows all 40 bits transmitted.

Data from humidity and temperature sensor.

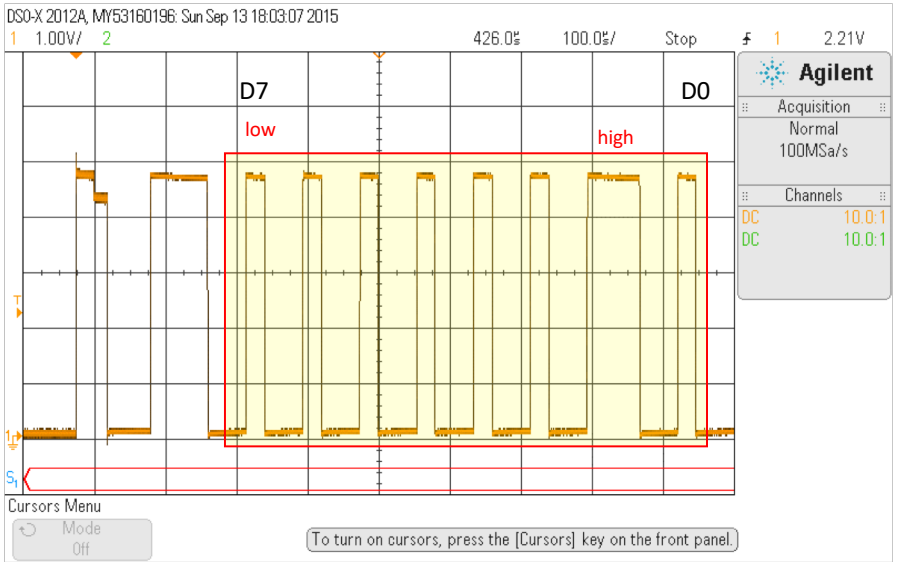
40 bits in total

32 bits of data
8 bit checksum.

16 bits for humidity

16 bits temperature

1st of 5 DHT22 Example: Humidity High Byte 0x02 70us (70) and 30 us (low)



Note: scope is set to 100 us /div.

If the bit has about 70 us high it represents a high.

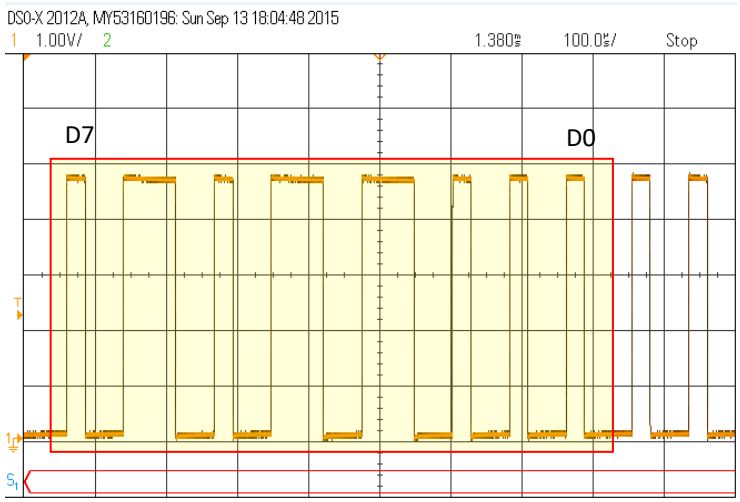
If the data is about 30 us high it is a low.

Humidity
High byte

00000010 = 0x02

0000 0010 0x02

2nd of 5 DHT22 Example: Humidity Low Byte 0x58



Humidity
Low byte

01011000 = 0x58

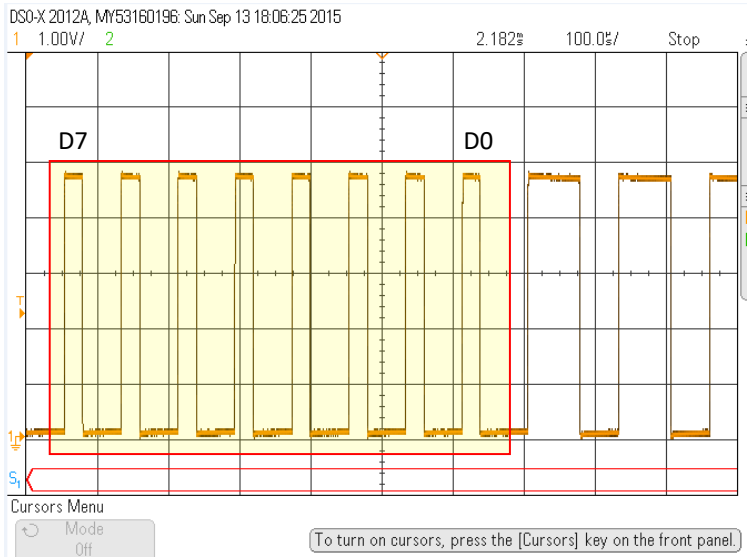
From high + low

0x0258 = 600d

Divide the value by 10 = 60%

0101 1000 0x58

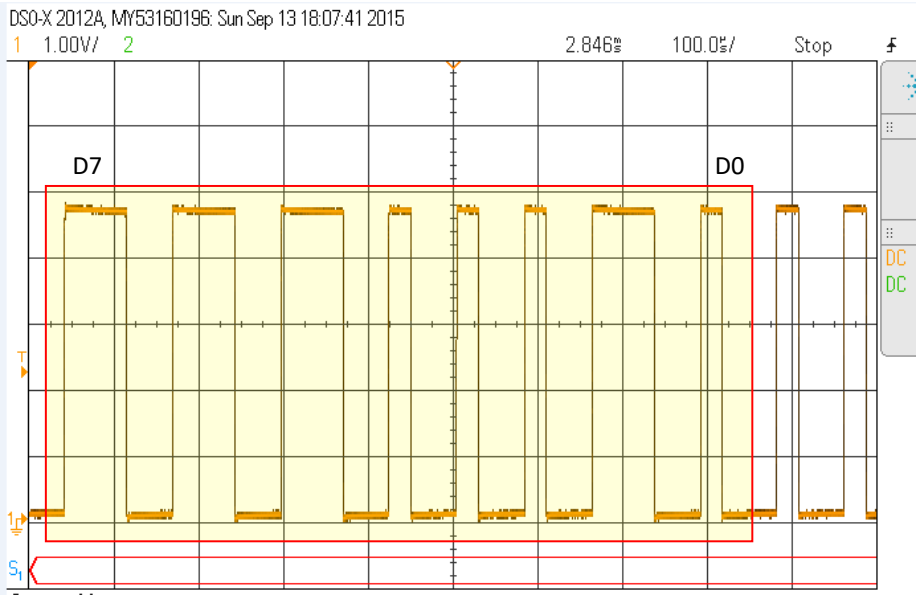
3rd of 5 DHT22 Example: Temperature High Byte 0x00



Temperature
High byte

00000000 = 0x00

4th of 5 DHT22 Example: Temperature Low Byte 0xE2



1110 0010 0xE2

Temperature
Low byte.

11100010 = 0xE2

High + low byte

= 0x00E2

= 226 / 10 = 22.6

I2C Eye 2 Sea -- Inter IC communications and the MCP4725 12 bit I2C DAC

Introduction

In this tutorial, you will learn all about the I^2C communication protocol, why you would want to use it, and how it's implemented.

The Inter-integrated Circuit (I^2C) Protocol is a protocol intended to allow multiple "slave" digital integrated circuits ("chips") to communicate with one or more "master" chips. Like the Serial Peripheral Interface (SPI), it is only intended for short distance communications within a single device. Like Asynchronous Serial Interfaces (such as RS-232 or UARTs), it only requires two signal wires to exchange information.

Devices that use I2C

Power Supply/LED Driver

- Control power-on/-off or dimming and configure mode of operation, output voltage, sequencing and slew rate for single- or multi-topology converters

Power Monitor

- Monitor current, voltage and average power, while minimizing software polling with min/max registers and configurable alerts

Temperature Monitor

- Measure combinations of voltage, current and internal or external temperature
- Trigger single or repeated measurements and change formats (Celsius or Kelvin)

Power over Ethernet Power Sourcing Equipment (PSE)

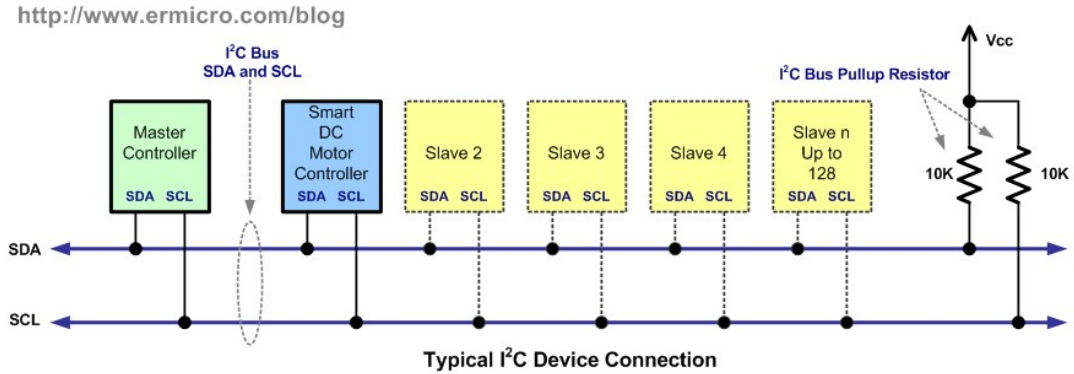
- Efficiently source up to 90W of power, while configuring PSE mode of operation and monitoring per port status, current, MOSFET health and die temperature

Battery Charger/Gas Gauge

- Adjust charge current, float voltage and charge termination, while monitoring status, charge, current, voltage or temperature of battery, USB or wall sources

ADC/DAC

- Write to or read from data converters with no latency, and select input or output data formats and use of internal or external reference



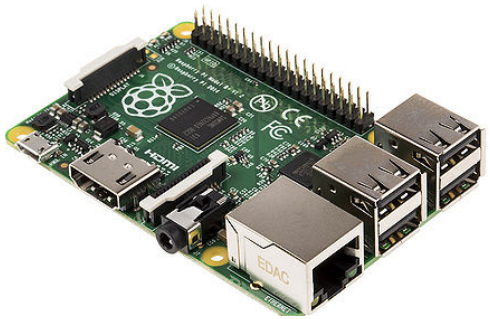
Data on the I2C bus can be transferred in three modes:

1) Standard Mode: 100kbps.
2) Fast Mode: 400kbps
3) High Speed Mode: 3.4Mbps.

The I2C bus uses Vcc, ground and 2 signal wires.

SDA – serial data
SCL – serial clock.

The bus is used to connect many devices to the I2C signals on the Arduino or other controllers such as a Rasberry PI low cost computer.



RaspberryPi

Type	Part #	Notes	Address	Address	Address
OLED	SSD1306	One addr pin	0x3C	0x3D	
NFC/RFID	PN532		0x48		
Sensor	TSL2561	1 tri-state addr pin	0x29	0x39	0x49
Sensor	BMP085	Can use XCLR to select/deselect	0x77		
Sensor	ADXL345	1 address pin	0x1D	0x53	
Sensor	HMC5883L		0x1E		
Sensor	BMA180		0x77		
Sensor	MMA7455L	Same as ADXL345	0x1D		
RTC	DS1307	Same as DS3231	0x68		
RTC	DS3231	Same as DS1307	0x68		
DAC	MCP4725A0	1 addr pin, 3 variants	0x60	0x61	
DAC	MCP4725A1	1 addr pin, 3 variants	0x62	0x63	
DAC	MCP4725A2	1 addr pin, 3 variants	0x64	0x65	

Each I2C device has a specific 7 bit address to select the I2C device.

In the lab three I2C devices will be used.

A MCP4725 12 bit DAC.

An ADXL345 Accelerometer.

A DS3231 real time clock.

There are many I2C devices that can be connected to the Arduino. Displays, pressure and temperature sensors, etc.

I²C Protocol MCP4725 Start, Stop, Address and Data and ACK

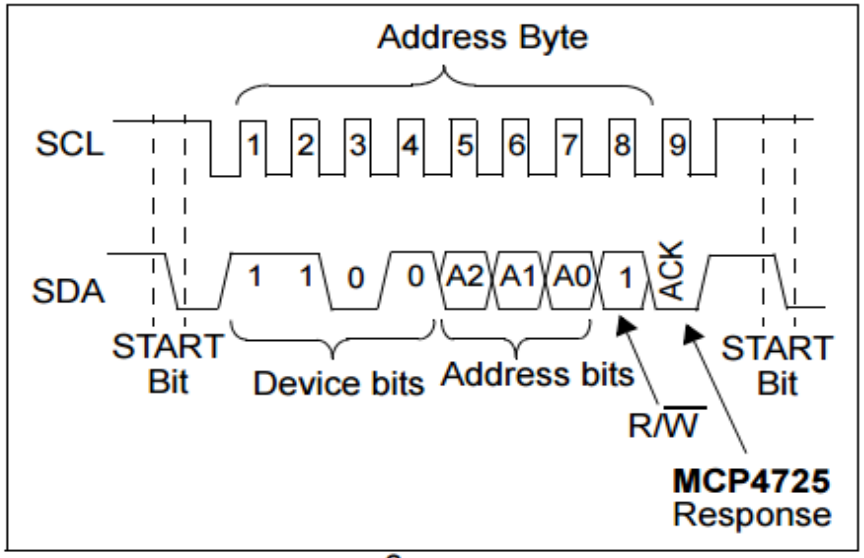


FIGURE 8-2: *I²C Bus Connection Test.*

Start – Clock is high when data changes to a low.

Stop – Clock is high when data goes high.

Stop/Start – only time data changes when clock is high. Normally data is stable when clock is high.

The clock signal is used to synchronize the data. The data is sampled and captured on the positive level of the clock signal. Each byte of data includes a start condition, a stop condition and an acknowledge bit that verifies the data transmission.

MCP4725 12 bit DAC Waveform

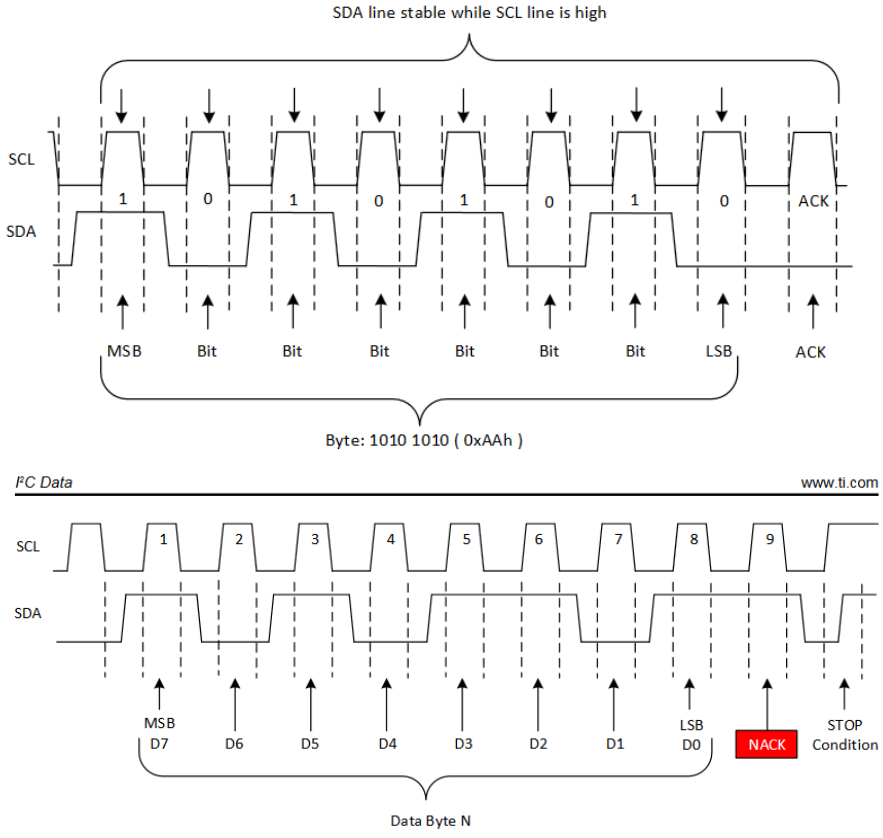


Figure 7. Example NACK Waveform

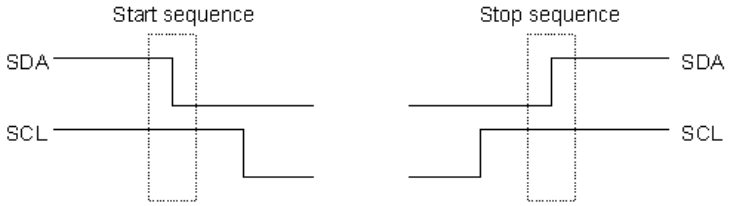
Data is sampled when clock is high.

First bit is MS.

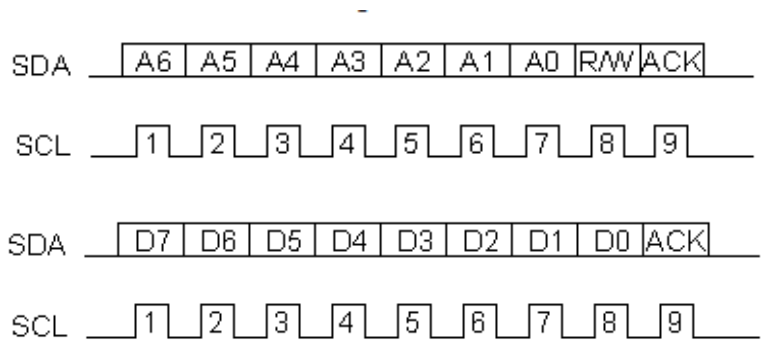
Bit 9 is the acknowledge bit.
0 is ok (ACK)
1 is not ok. (NACK)

MCP4725 12 bit DAC Waveform

www.robot-electronics.co.uk/i2c-tutorial



The start sequence and stop sequence are special in that these are the only places where the SDA (data line) is allowed to change while the SCL (clock line) is high. When data is being transferred, SDA must remain stable and not change whilst SCL is high. The start and stop sequences mark the beginning and end of a transaction with the slave device.



Device Address

Device Data

MCP4725 12 bit DAC Waveform

www.ti.com/lit/an/slva704/slva704.pdf

2.1.1 START and STOP Conditions

I²C communication with this device is initiated by the master sending a START condition and terminated by the master sending a STOP condition. A high-to-low transition on the SDA line while the SCL is high defines a START condition. A low-to-high transition on the SDA line while the SCL is high defines a STOP condition.

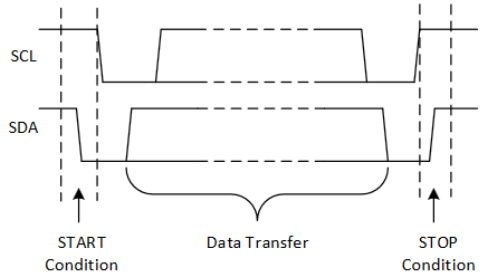


Figure 5. Example of START and STOP Condition

Write to One Register in a Device

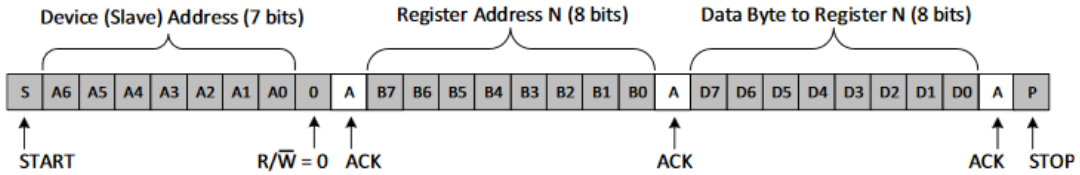


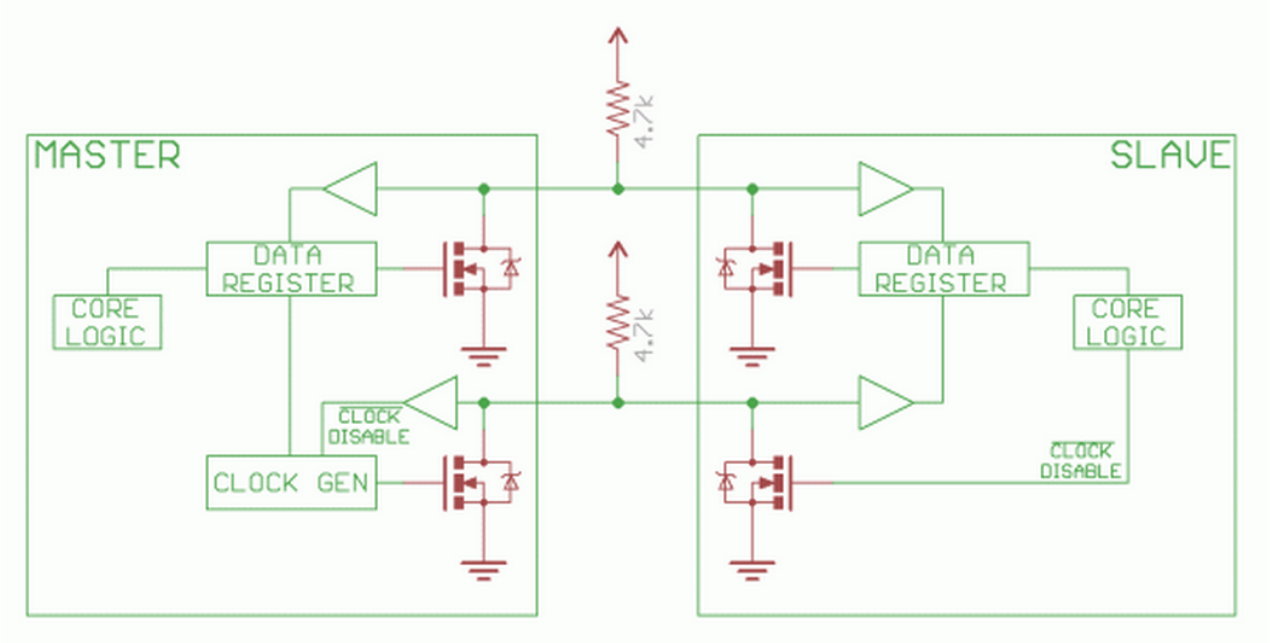
Figure 8. Example I²C Write to Slave Device's Register

I2C at the Hardware Level

Signals

Each I^2C bus consists of two signals: SCL and SDA. SCL is the clock signal, and SDA is the data signal. The clock signal is always generated by the current bus master; some slave devices may force the clock low at times to delay the master sending more data (or to require more time to prepare data before the master attempts to clock it out). This is called “clock stretching” and is described on the protocol page.

Unlike UART or SPI connections, the I^2C bus drivers are “open drain”, meaning that they can pull the corresponding signal line low, but cannot drive it high. Thus, there can be no bus contention where one device is trying to drive the line high while another tries to pull it low, eliminating the potential for damage to the drivers or excessive power dissipation in the system. Each signal line has a pull-up resistor on it, to restore the signal to high when no device is asserting it low.



Notice the two pull-up resistors on the two communication lines.

Resistor selection varies with devices on the bus, but a good rule of thumb is to start with 4.7k and adjust down if necessary. I^2C is a fairly robust protocol, and can be used with short runs of wire (2-3m). For long runs, or systems with lots of devices, smaller resistors are better.

Signal Levels

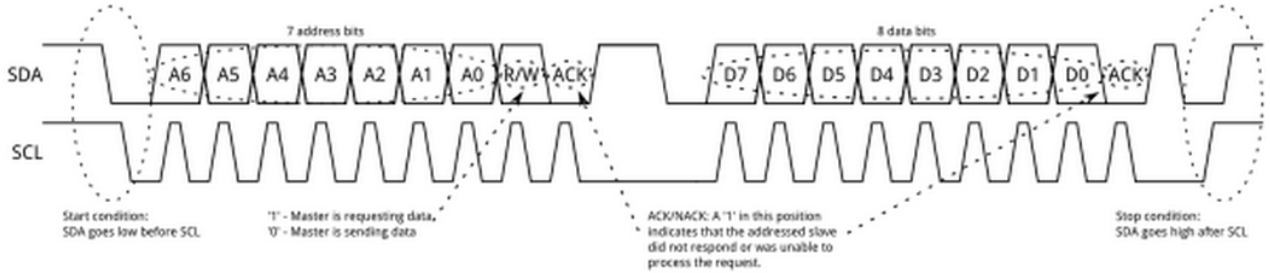
Since the devices on the bus don't actually drive the signals high, I^{2}C allows for some flexibility in connecting devices with different I/O voltages. In general, in a system where one device is at a higher voltage than another, it may be possible to connect the two devices via I^{2}C without any level shifting circuitry in between them. The trick is to connect the pull-up resistors to the lower of the two voltages. This only works in some cases, where the lower of the two system voltages exceeds the high-level input voltage of the the higher voltage system–for example, a 5V Arduino and a 3.3V accelerometer.

If the voltage difference between the two systems is too great (say, 5V and 2.5V), SparkFun offers a simple I^{2}C level shifter board. Since the board also includes an enable line, it can be used to disable communications to selected devices. This is useful in cases where more than one device with the same address is to be connected to a single master–Wii Nunchucks are a good example.

Protocol

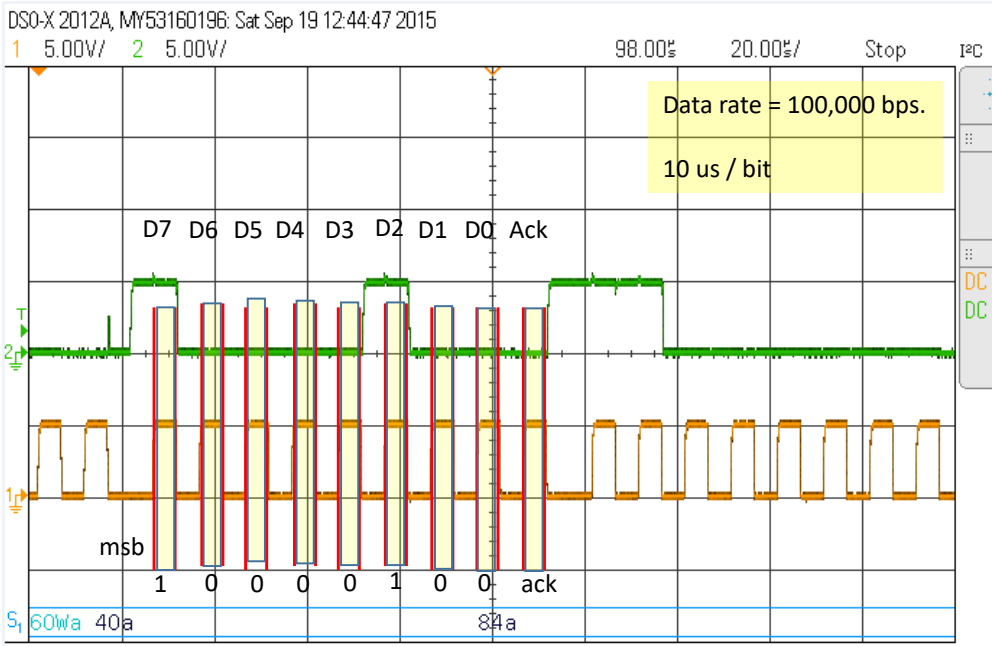
Communication via I^2C is more complex than with a UART or SPI solution. The signalling must adhere to a certain protocol for the devices on the bus to recognize it as valid I^2C communications. Fortunately, most devices take care of all the fiddly details for you, allowing you to concentrate on the data you wish to exchange.

Basics



Messages are broken up into two types of frame: an address frame, where the master indicates the slave to which the message is being sent, and one or more data frames, which are 8-bit data messages passed from master to slave or vice versa. Data is placed on the SDA line after SCL goes low, and is sampled after the SCL line goes high. The time between clock edge and data read/write is defined by the devices on the bus and will vary from chip to chip.

MCP4725 12 bit DAC Waveform



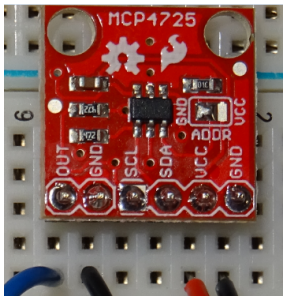
MCP4725 I2C DAC

In this example 8 bits of data

10000100 are transmitted. MSB is first.

MCP4725 12 bit DAC (Arduino sketch)

```
10 #include <Wire.h>//Include the Wire library to talk I2C
11
12 //This is the I2C Address of the MCP4725, by default (A0 pulled to GND).
13
14 #define MCP4725_ADDR 0x60   // address 0x60
15 //For devices with A0 pulled HIGH, use 0x61
16
17 void setup()
18 {
19     Wire.begin(); // start 2 wire IIC
20 }
21 void loop()  // runs endlessly
22 {
23
24   Wire.beginTransmission(MCP4725_ADDR); // 12 bit DAC
25   Wire.write(0x40); //  cmd to update the DAC
26   Wire.write(0x7A); // the 8 most significant bits...
27   Wire.write(0xD0); // the 4 least significant bits...
28   Wire.endTransmission();
29   delay(3000);
30 }
```



MCP4725 I2C DAC
Data = 0x7AD
First 12 bits of the two byte number

= 1965 decimal

1965/4096
= 0.4797

0.4797 * 5.0 volts
= 2.4 volts.

MCP4725 12 bit DAC (Arduino sketch)

```
10 #include <Wire.h>//Include the Wire library to talk I2C
11
12 //This is the I2C Address of the MCP4725, by default (A0 pulled to GND).
13
14 #define MCP4725_ADDR 0x60   // address 0x60
15 //For devices with A0 pulled HIGH, use 0x61
16
17 void setup()
18 {
19     Wire.begin(); // start 2 wire IIC
20 }
21 void loop()  // runs endlessly
22 {
23
24   Wire.beginTransmission(MCP4725_ADDR); // 12 bit DAC
25   Wire.write(0x40); //  cmd to update the DAC
26   Wire.write(0x90); // the 8 most significant bits...
27   Wire.write(0x00); // the 4 least significant bits...
28   Wire.endTransmission();
29   delay(3000);
30 }
```

MCP4725 I2C DAC
Data = 0x900

= 2304 decimal

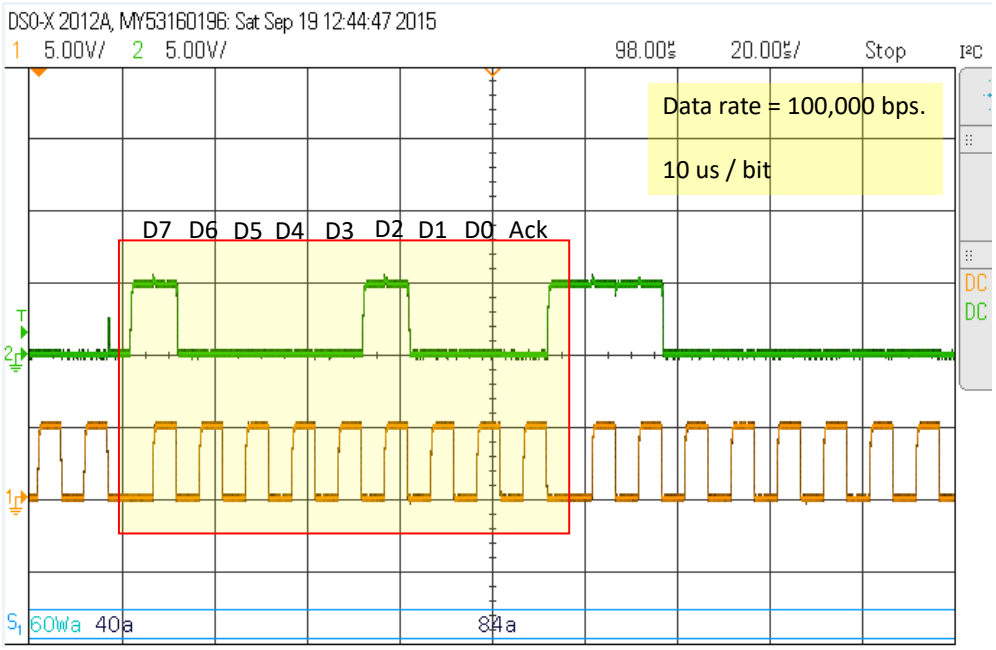
2304/4096
= 0.5625

0.5625 * 5.0 volts
= 2.8 volts.

MCP4725 Addr = 0x60 | MCP4725 Command 0x40 | DAC value upper 8 bits | DAC low 4 bits

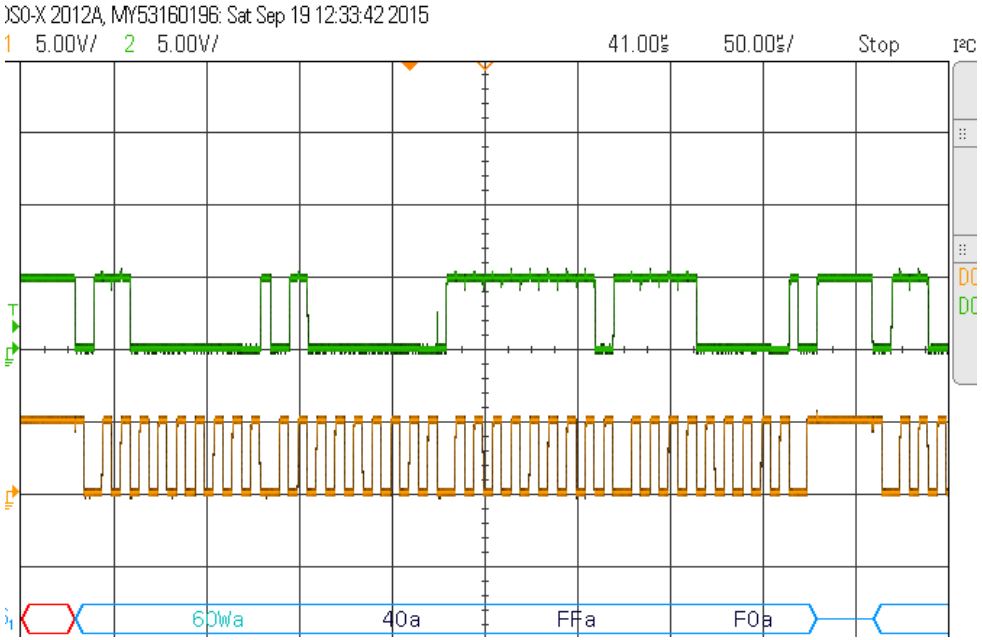
The DAC Data is transmitted as two bytes (4 nibbles). The first byte represent the 1st 8 bits of the data. The next byte (upper 4 bits) represents the lower 4 bits of the data. There are 12 bits in the data. The MCP4725 address is 0x60.

MCP4725 12 bit DAC Waveform



MCP4725 I2C DAC
Data = 84

MCP4725 12 bit DAC Waveform

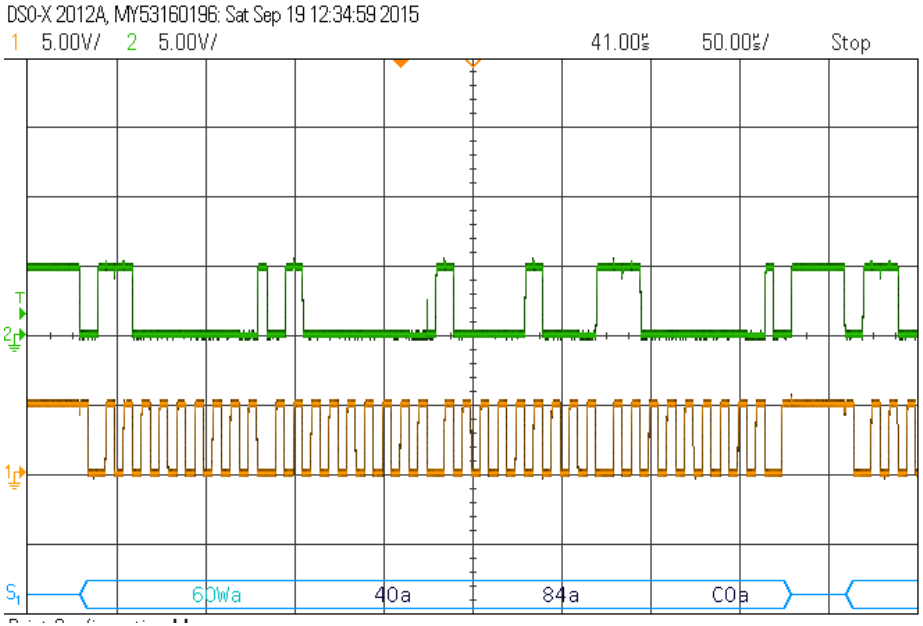


MCP4725 I2C DAC
Data = FFF

FFF/FFF * 5.0 volts

= 5.0

MCP4725 12 bit DAC Waveform



MCP4725 I2C DAC
Data = 0x84C

0x84C = 2124 $_{dec}$

2124/4096 * 5.0 volts

= 2.6 volts.

Asynchronous Serial I/O and RS-232

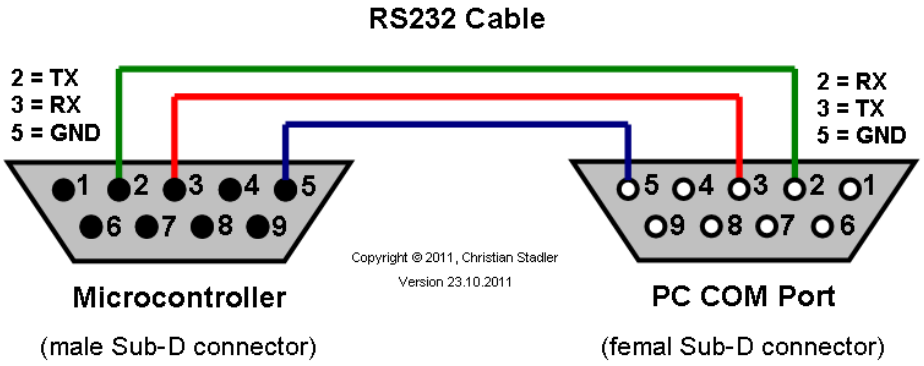
ASCII TABLE

American Standard Code for Information Interchange

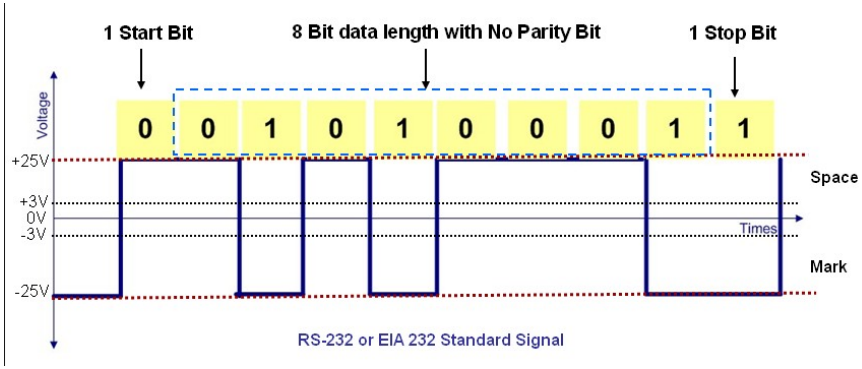
Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

The ASCII table shows the standard code used to transfer characters.

Examples: “1” = 0x31, “A” = 0x41 “U” = 0x55 or 01010101 in binary



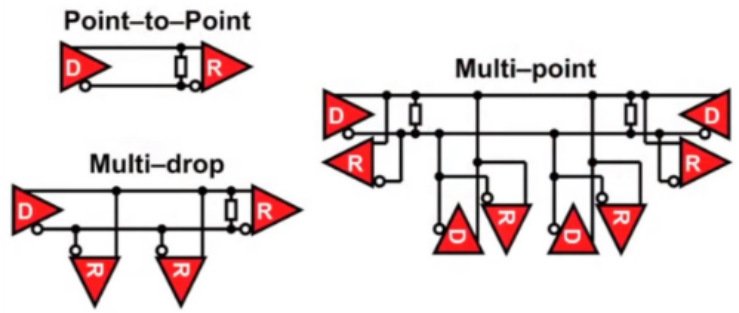
An RS-232 serial connector has 9 pins. At least 3 pins are required to transmit and receive data



At TTL level the data begins with a low start bit, 8 data bits and a high stop bit.

To transmit at greater distance RS-232 levels are used. Typical values are +/- 12 volts.

Different Bus topolgy
Point to Point - consist of one driver and one reciever
Multi Drop - One driver with Multiple reciever
Multi Point - Multiple drivers and Multiple Recievers

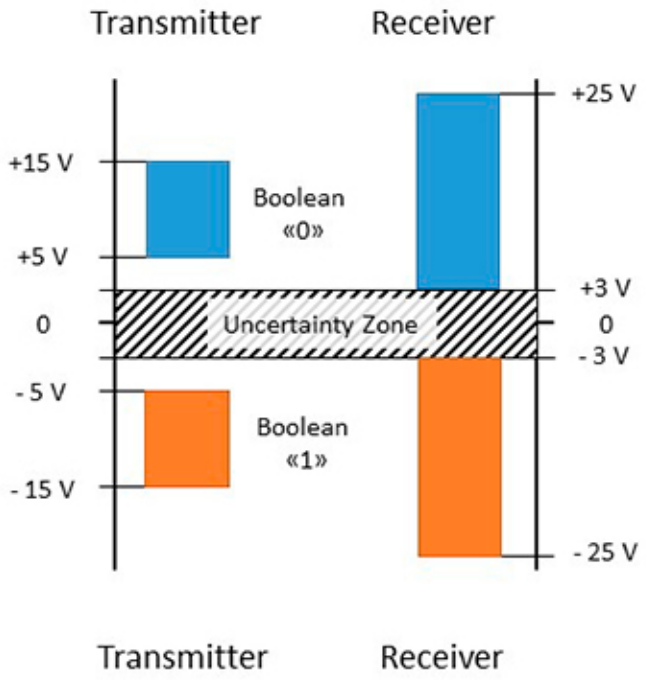


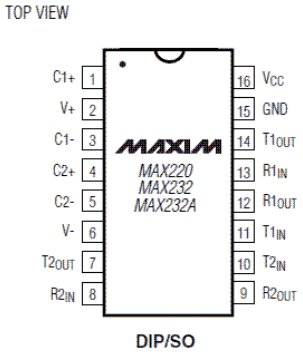
RS 232 Transmission is
Single Ended Point to Point Bus toplogy
At Driver End
Logic 0 - 5v to 15v
Logic 1 - 5v to -15v

At Reciever End
Logic 0 +3 to 15v
Logic 1 -3 to -15v

RS 485 is

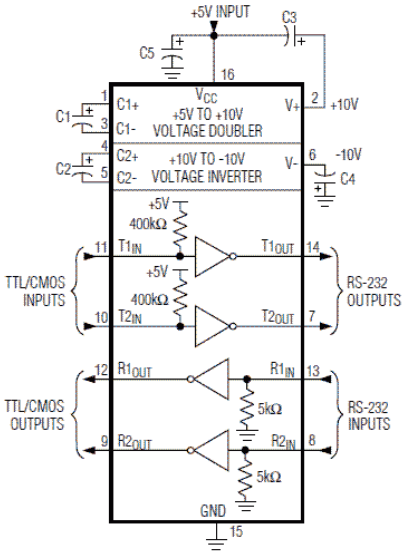
Differential Multi Point interface
Up to 32 devices can be connected

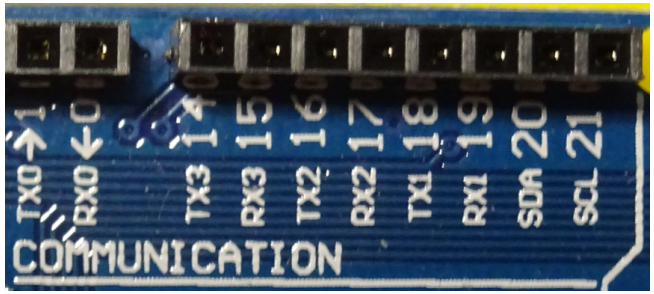




CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	0.047	0.33	0.33	0.33	0.33
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Diagrams continued in the full data sheet.





To convert from TTL levels to RS-232 levels it is common to use a level translator called a MAX232. This chip is manufactured by MAXIM.

The chip converts a logic high (5 volts) to (-12 volts)

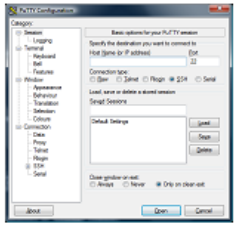
A logic low (0 volts) is converted to (+12 volts DC)

Baud	Bit Width	Real Data Rate * (1 start bit, 8 data bits, no parity, 1 stop bit)
4800	208.33us	480 Bytes/s 0.47 KBytes/s
9600	104.17us	960 Bytes/s 0.94 KBytes/s
19200	52.08us	1920 Bytes/s 1.88 KBytes/s
38400	26.04us	3840 Bytes/s 3.75 KBytes/s
57600	17.36us	5760 Bytes/s 5.63 KBytes/s
115200	8.68us	11520 Bytes/s 11.25 KBytes/s

Standard asynchronous serial bit rates.

A typical USB to RS-232 converter.

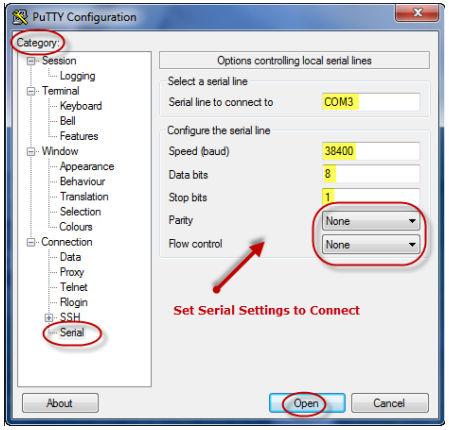
Putty is a common (free) serial terminal program for a PC or MAC computer. It can be used to receive and transmit serial data from the Arduino or myDAQ to a PC or MAC.

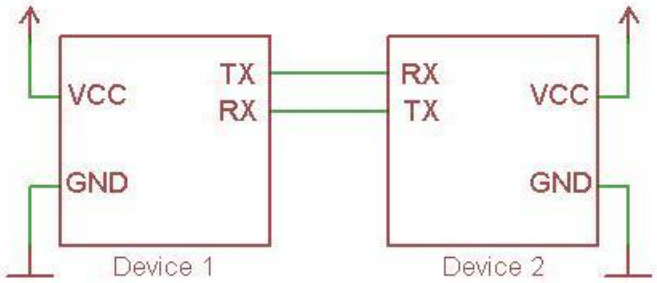


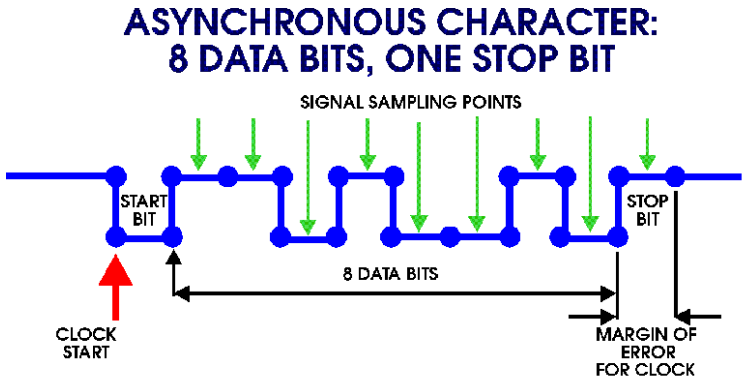
Download PuTTY

PuTTY is an SSH and telnet client, by a group of volunteers.

You can download PuTTY here.







Many devices used in the electrical industry transmit and receive data using Asynchronous serial transmission. This method of communications requires only 3 wires, TXD, RXD and ground. Data can transmit at rates from 9600 bps to

The receiver and transmitter must operate at the same data rate. One of the more common data rates is **9600 bps**. Other "standard" bit/second rates are 1200, 2400, 4800, 19200, 38400, 57600, and 115200.

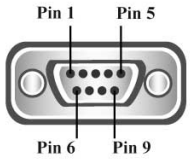
At 9600 bits/second one data bit takes 104 us.

Start and stop bits are used for synchronization.

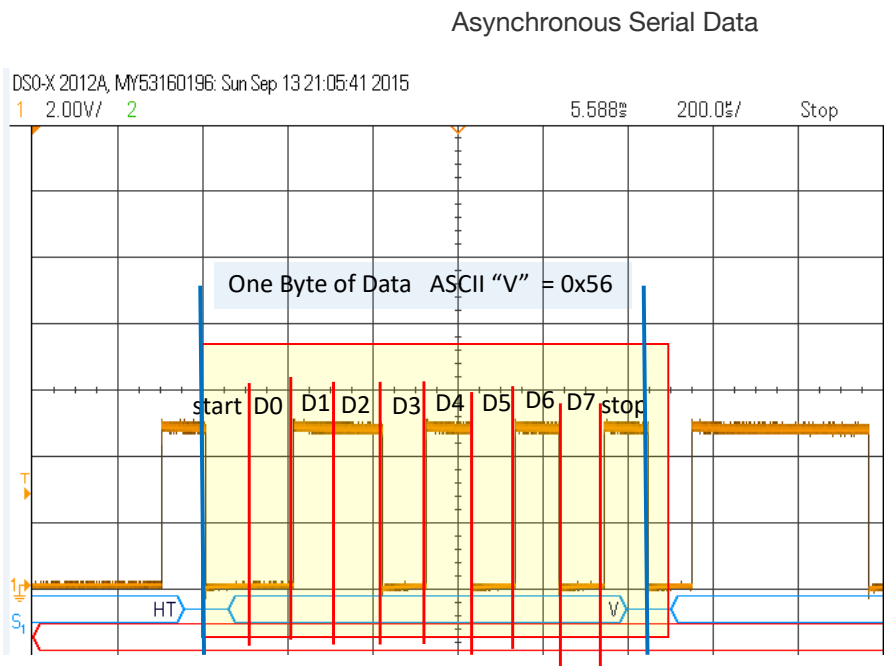
RS232

Pin 1	DCD
Pin 2	RXD
Pin 3	TXD
Pin 4	DTR
Pin 5	GND
Pin 6	DSR
Pin 7	RTS
Pin 8	CTS
Pin 9	RI

RS232 Pinout (9 Pin Male)



Asynchronous Serial Data Waveform



Start bit is always low, stop bit is always high.

1 byte transmits in 1.06 ms.

That is about 1000 bytes per second.

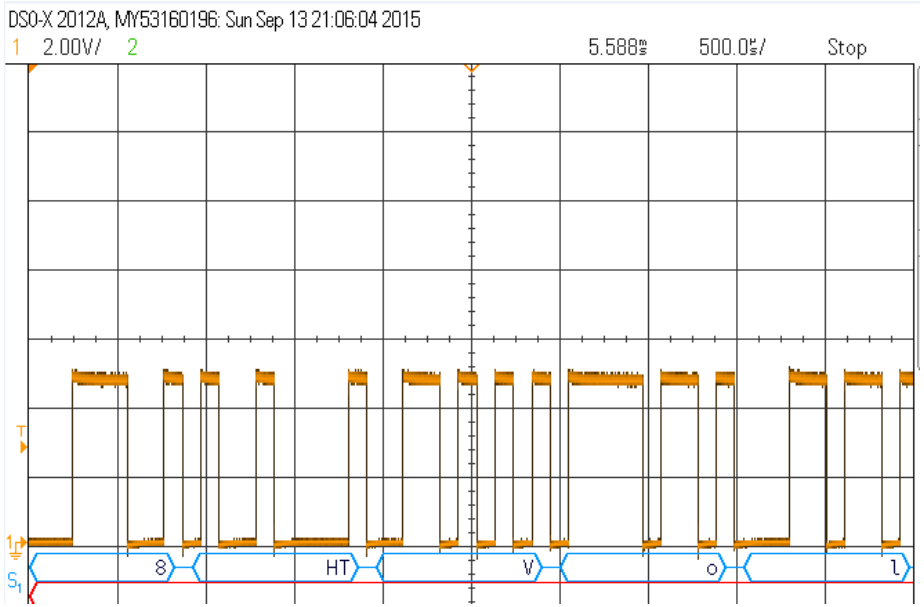
Not fast enough to send a video but more than fast enough to transmit data such as temperature and humidity.

01010110 = 0x56 = "V"

This is an example of the ASCII code "V" being transmitted.

Asynchronous Serial Data Waveform

Asynchronous Serial Data -- Example



This is an example of a message being transmitted. 8, horizontal tab, V,o,l

Arduino and Logic Gate Voltage Parameters

Voltage Parameters

V_{OH} (min) - High Level Output Voltage

This is the minimum voltage level for a logic 1, at the output of the gate, under load.

V_{OL} (max) - Low Level Output Voltage

This is the maximum voltage level for a logic 0, at the output of the gate, under load.

Voltage Parameters

V_{IH} (min) - High Level Input Voltage

This is the minimum voltage level required at an input for a logic 1.

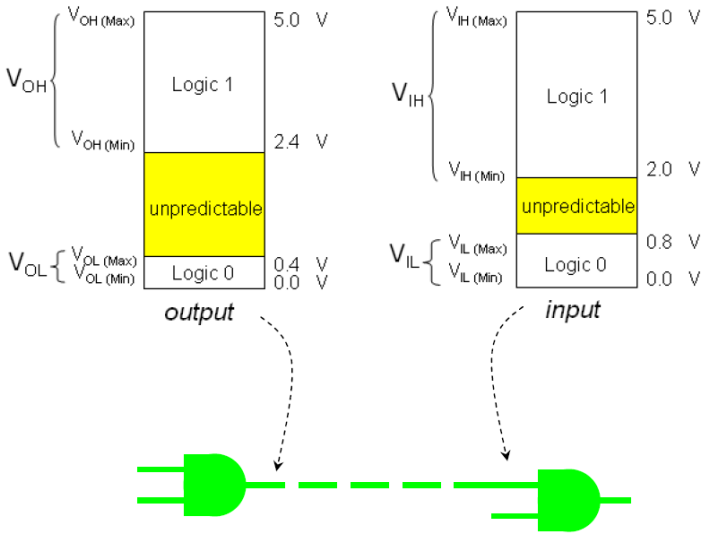
If the input voltage falls below this point, it will not be considered a logic 1.

V_{IL} (max) - Low Level Input Voltage

This is the maximum voltage level required at an input for a logic 0.

If the input voltage rises above this point, it will not be considered a logic 0.

TTL Logic Levels



VOH – the minimum output voltage from the Gate.

VOL – The maximum gate output voltage.

VIH – The minimum input voltage for the gate.

VIL – The maximum input voltage for the gate.

If the voltages do not meet the minimum or maximum values indicated, the device may not read the voltage as a proper logic high or low.

When connecting two digital devices you must verify that the input and output voltages will meet the specifications of the two devices. You must check the VOH, VOL, VIH, VIL spec.

Arduino DC Electrical Specs:

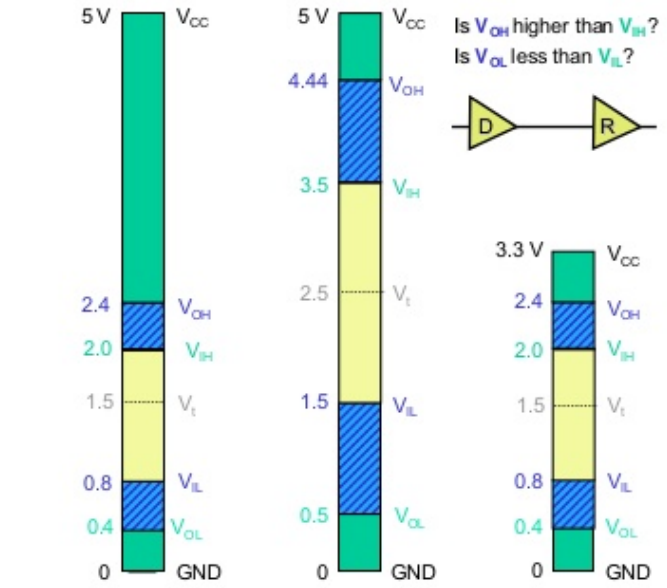
T_A = -40°C to 85°C, V_{CC} = 1.8V to 5.5V (unless otherwise noted)

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
V_{IL}	Input Low Voltage, Except XTAL1 and Reset pin	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	-0.5 -0.5		$0.2V_{CC}$[1] $0.3V_{CC}$[1]	V
V_{IL1}	Input Low Voltage, XTAL1 pin	V_{CC} = 1.8V - 5.5V	-0.5		$0.1V_{CC}$[1]	
V_{IL2}	Input Low Voltage, RESET pin	V_{CC} = 1.8V - 5.5V	-0.5		$0.1V_{CC}$[1]	
V_{IH}	Input High Voltage, Except XTAL1 and RESET pins	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.7V_{CC}$[2] $0.6V_{CC}$[2]		V_{CC} + 0.5 V_{CC} + 0.5	
V_{IH1}	Input High Voltage, XTAL1 pin	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.8V_{CC}$[2] $0.7V_{CC}$[2]		V_{CC} + 0.5 V_{CC} + 0.5	
V_{IH2}	Input High Voltage, RESET pin	V_{CC} = 1.8V - 5.5V	$0.9V_{CC}$[2]		V_{CC} + 0.5	
V_{OL}	Output Low Voltage[3], Except RESET pin	I_{OL} = 20mA, V_{CC} = 5V I_{OL} = 10mA, V_{CC} = 3V			0.9 0.6	
V_{OH}	Output High Voltage[4], Except RESET pin	I_{OH} = -20mA, V_{CC} = 5V I_{OH} = -10mA, V_{CC} = 3V	4.2 2.3			
I_{IL}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin low (absolute value)			1	µA
I_{IH}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin high (absolute value)			1	
R_{RST}	Reset Pull-up Resistor		30		60	kΩ
R_{PU}	I/O Pin Pull-up Resistor		20		50	

VIH, VIL, VOH, VOL DC specs. For the Arduino

IC Basics

Comparison of Switching Standards



5-V TTL
Standard TTL: ABT, AHCT, HCT, ACT, Bipolar

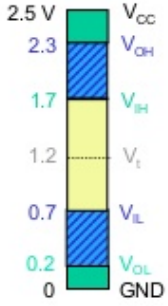
5-V CMOS
Rail-to-Rail 5 V HC, AHC, AC, LV-A

Is V_{OH} higher than V_{IH}?
Is V_{OL} less than V_{IL}?

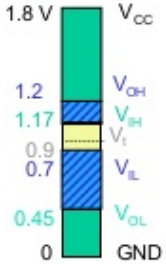
3.3-V LVTTL
LVT, LVC, ALVC AUP, LV-A, ALVT

D \ R	5TTL	5CMOS	3LVTTL	2.5CMOS	1.8CMOS
5TTL	Yes	No	Yes *	Yes*	Yes*
5 CMOS	Yes	Yes	Yes*	Yes*	Yes*
3 LVTTL	Yes	No	Yes	Yes*	Yes*
2.5 CMOS	Yes	No	Yes	Yes	Yes*
1.8 CMOS	No	No	No	No	Yes

* Requires V_{IH} Tolerance



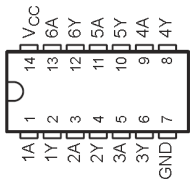
2.5-V CMOS
AUC, AUP, AVC, ALVC, LVC, ALVT



1.8-V CMOS
AUC, AUP, AVC, ALVC, LVC

Texas Instruments

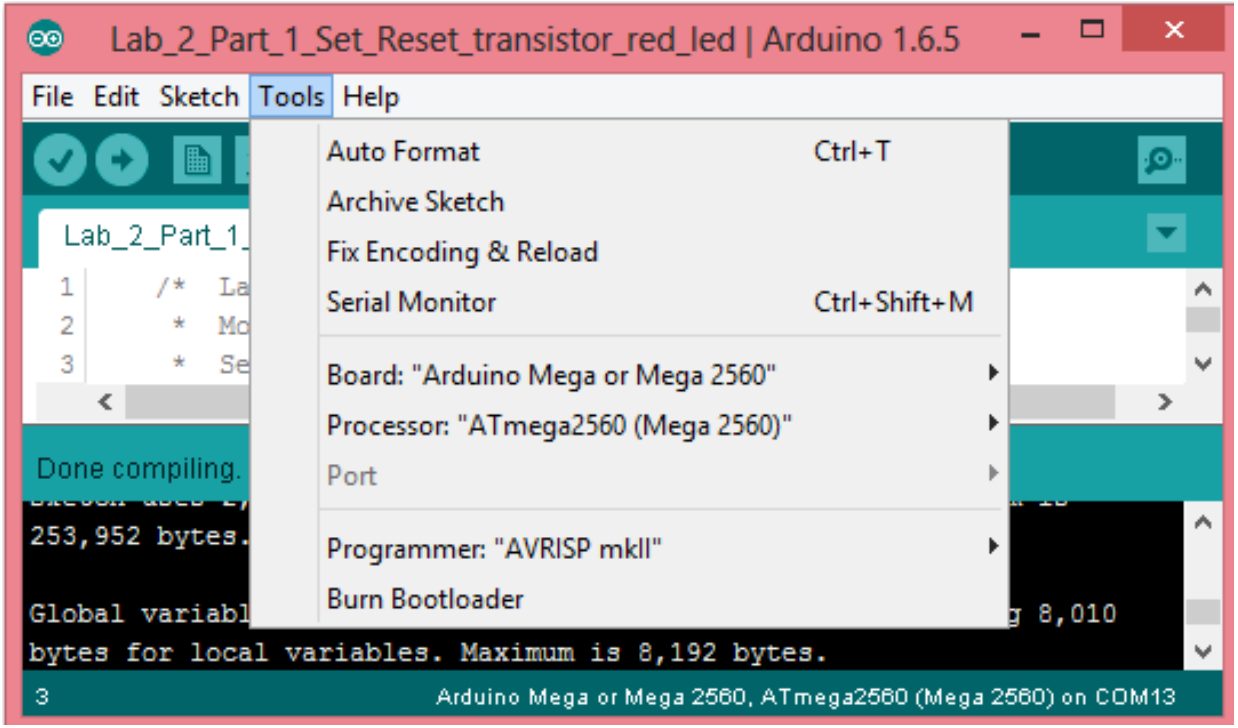
Symbol	Parameter	Condition	V_{CC} (V)	−55 to 25°C
V_{IH}	Minimum High-Level Input Voltage	V_{out} = 0.1V or V_{CC} −0.1V $\|I_{out}\| \leq 20\mu A$	2.0 3.0 4.5 6.0	1.50 2.10 3.15 4.20
V_{IL}	Maximum Low-Level Input Voltage	V_{out} = 0.1V or V_{CC} − 0.1V $\|I_{out}\| \leq 20\mu A$	2.0 3.0 4.5 6.0	0.50 0.90 1.35 1.80
V_{OH}	Minimum High-Level Output Voltage	$V_{in} = V_{IH}$ or V_{IL} $\|I_{out}\| \leq 20\mu A$	2.0 4.5 6.0	1.9 4.4 5.9
		$V_{in} = V_{IH}$ or V_{IL} $\|I_{out}\| \leq 2.4mA$ $\|I_{out}\| \leq 4.0mA$ $\|I_{out}\| \leq 5.2mA$	3.0 4.5 6.0	2.48 3.98 5.48
V_{OL}	Maximum Low-Level Output Voltage	$V_{in} = V_{IH}$ or V_{IL} $\|I_{out}\| \leq 20\mu A$	2.0 4.5 6.0	0.1 0.1 0.1
		$V_{in} = V_{IH}$ or V_{IL} $\|I_{out}\| \leq 2.4mA$ $\|I_{out}\| \leq 4.0mA$ $\|I_{out}\| \leq 5.2mA$	3.0 4.5 6.0	0.26 0.26 0.26



74HC04; 74HCT04
Hex inverter
Rev. 4 — 3 August 2012 Product data sheet

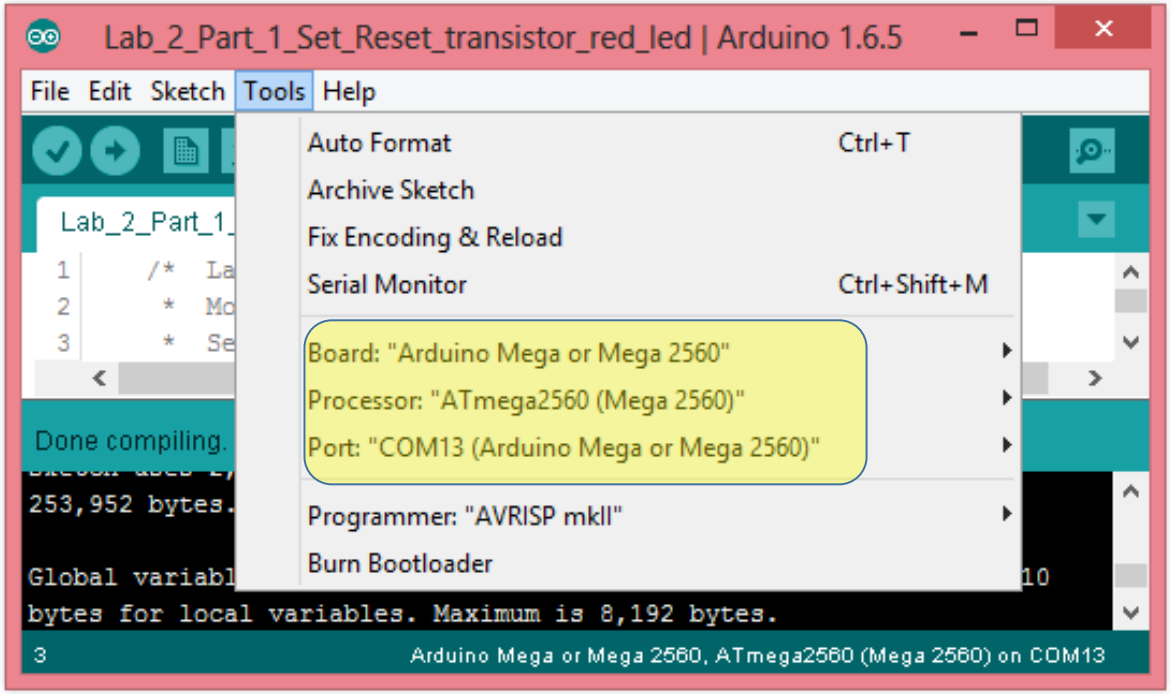
VIH, VIL, VOH, VOL DC specs. For the 74HC04

Arduino Software

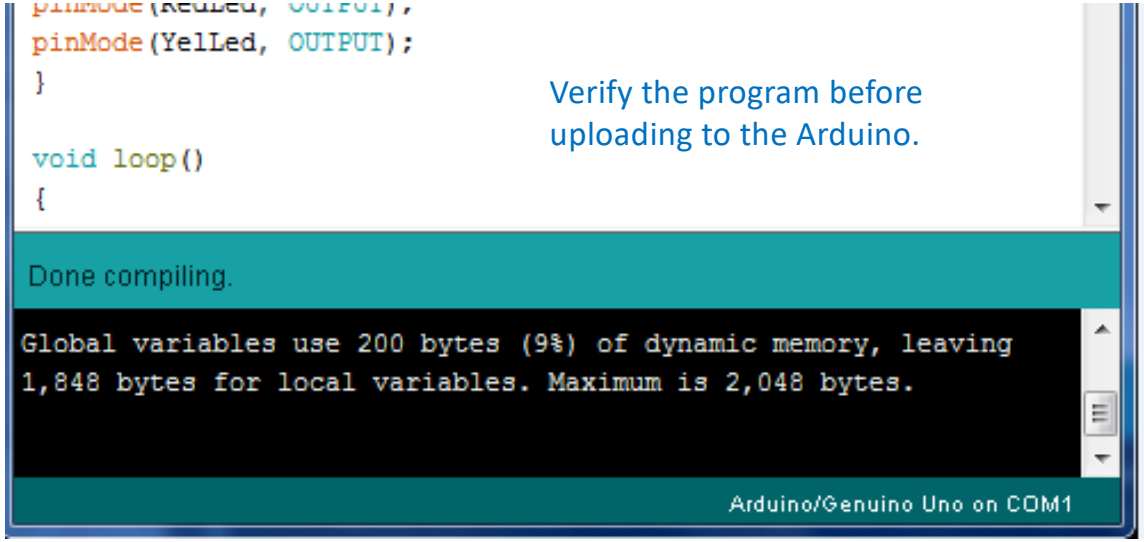


Under the Tools tab verify that the board type and Port are properly configured. What is the problem with the setup?

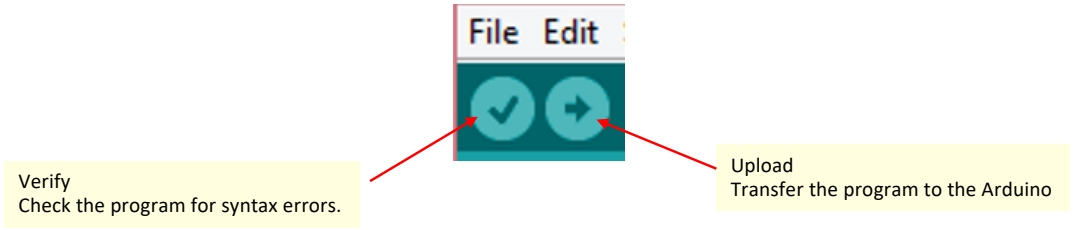
Device not connected. No port.



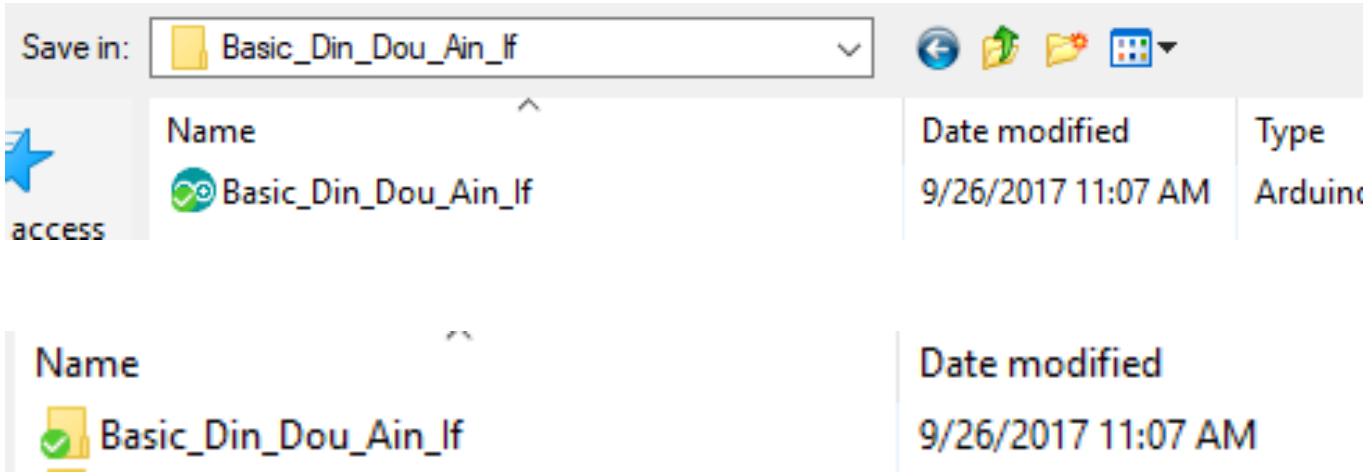
Proper Configuration Board, Processor and Port Match:

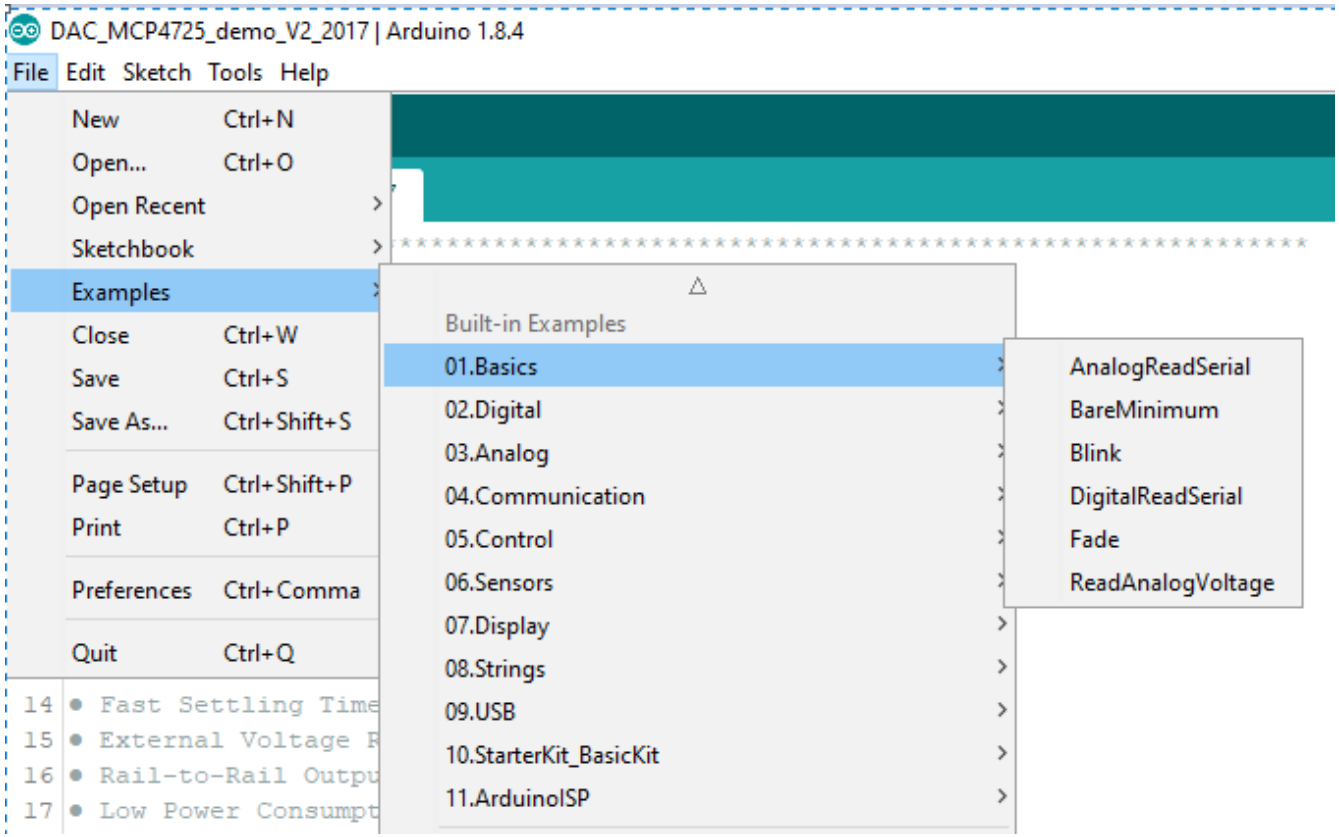


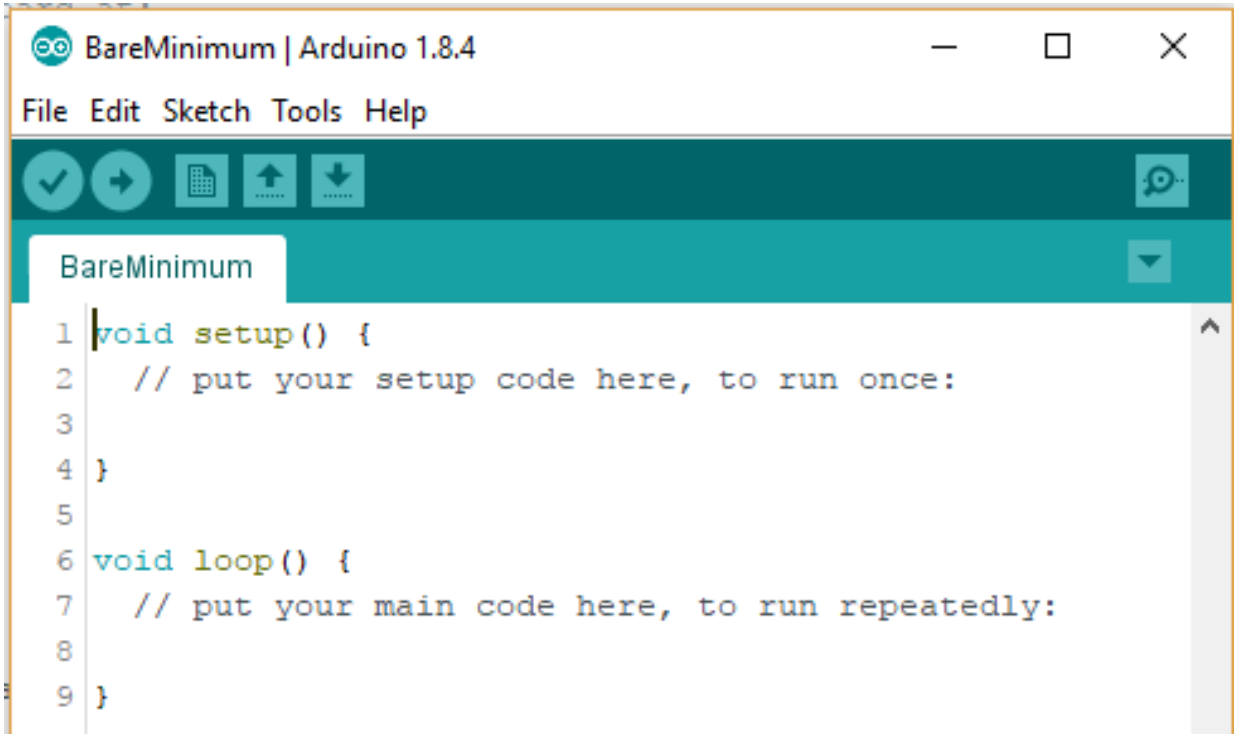
An Arduino Mega is connected to COM8. What is the problem with the setup?

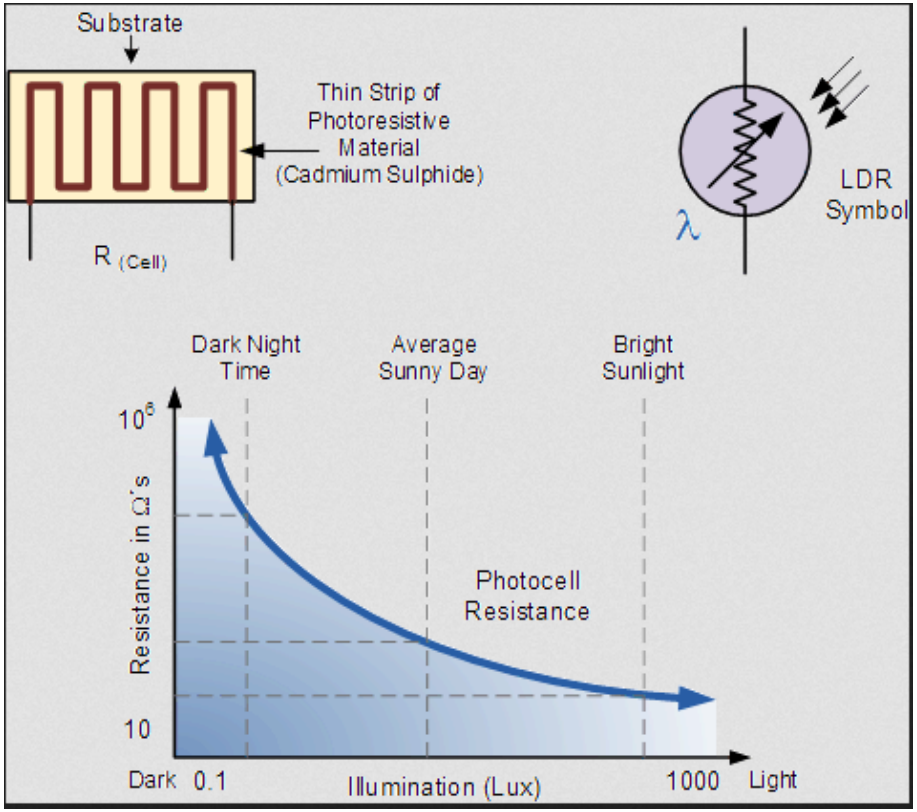


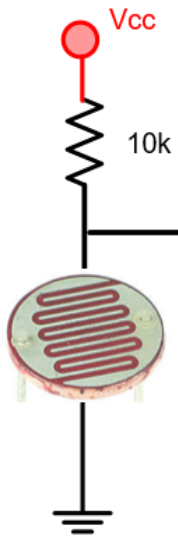
Arduino.ino file must be inside a folder of the same name.
The first save creates the folder.




DAC_MCP4725_demo_V2_2017 | Arduino 1.8.4
File Edit Sketch Tools Help
New Ctrl+N
Open... Ctrl+O
Open Recent
Sketchbook
Examples
Close Ctrl+W
Save Ctrl+S
Save As... Ctrl+Shift+S
Page Setup Ctrl+Shift+P
Print Ctrl+P
Preferences Ctrl+Comma
Quit Ctrl+Q
Built-in Examples
01.Basics
02.Digital
03.Analog
04.Communication
05.Control
06.Sensors
07.Display
08.Strings
09.USB
10.StarterKit_BasicKit
11.ArduinoISP
AnalogReadSerial
BareMinimum
Blink
DigitalReadSerial
Fade
ReadAnalogVoltage
14 • Fast Settling Time
15 • External Voltage R
16 • Rail-to-Rail Outpu
17 • Low Power Consumpt


BareMinimum | Arduino 1.8.4
File Edit Sketch Tools Help
BareMinimum
1 void setup() {
2 // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7 // put your main code here, to run repeatedly:
8
9 }





Lower voltage when brighter.

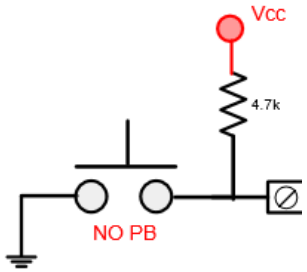
~ 500 ohms outdoors sunny (0.24 v)
~ 6,000 ohms room light (1.9 volts)
~ 50,000 ohms in darkness (4.2 volts)

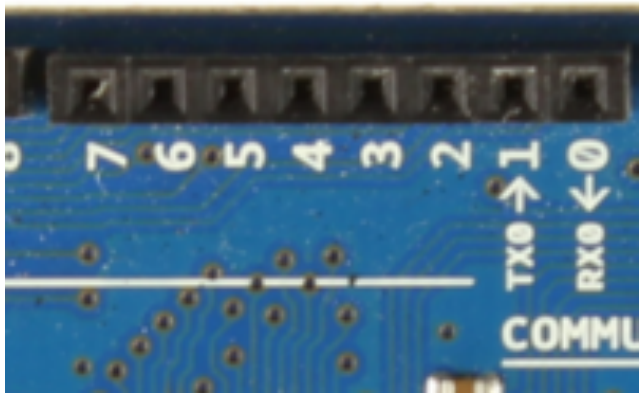
Lab_2_Main_Program_CAM8302E_2017_V2 | Arduino 1.8.4

File Edit Sketch Tools Help

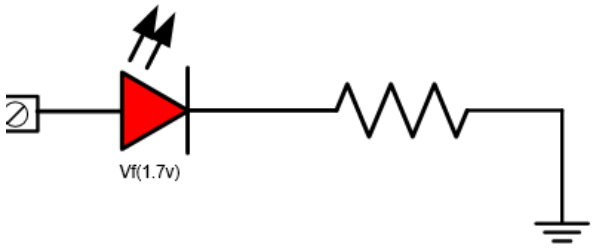
Lab_2_Main_Program_CAM8302E_2017_V2

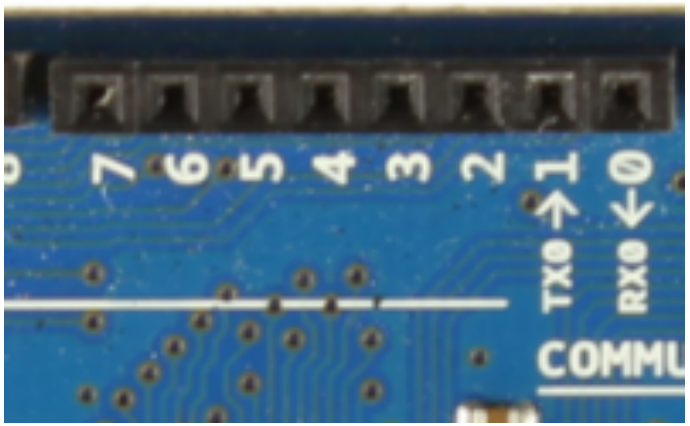
```
1   /*  Lab_2_F2017 */
2
3   /* Michel Hanbury CAM8302E Fall 2017 September 25, 2017 */
4   /* Variables are case sensitive */
5
6   #define  Red_Led  6
7   #define  Yel_Led  5
8   #define Grn_Led  4
9   #define Clear_Warm  3
10  #define Clear_Hot  2
11
```





```
13    void setup()
14    {
15    Serial.begin(9600);     // configure the serial port for 9600 bits per second (104 us/bit
16
17    pinMode(Red_Led,OUTPUT);     // output on pin 6
18    pinMode(Yel_Led,OUTPUT);     // output on pin 5
19    pinMode(Grn_Led,OUTPUT);
20    pinMode(Clear_Warm,INPUT);
21    pinMode(Clear_Hot,INPUT);    // push button input on pin 2
22    digitalWrite(Grn_Led,HIGH);
23    }
```

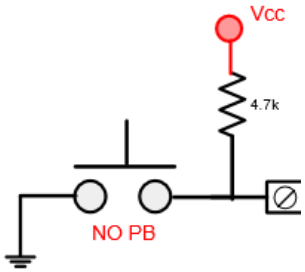




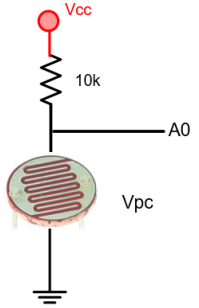
```
44      if (!digitalRead(Clear_Hot)and (val > 220))
45      {
46        digitalWrite(Red_Led,LOW);
47        digitalWrite(Yel_Led,LOW);
48
49      }
```

```
26        void loop()
27      {
28      boolean sw1;
29      int val = analogRead(A0);
30      sw1 = digitalRead(3);
31      if (!sw1 and (val > 220))
```

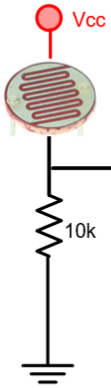
2nd approach



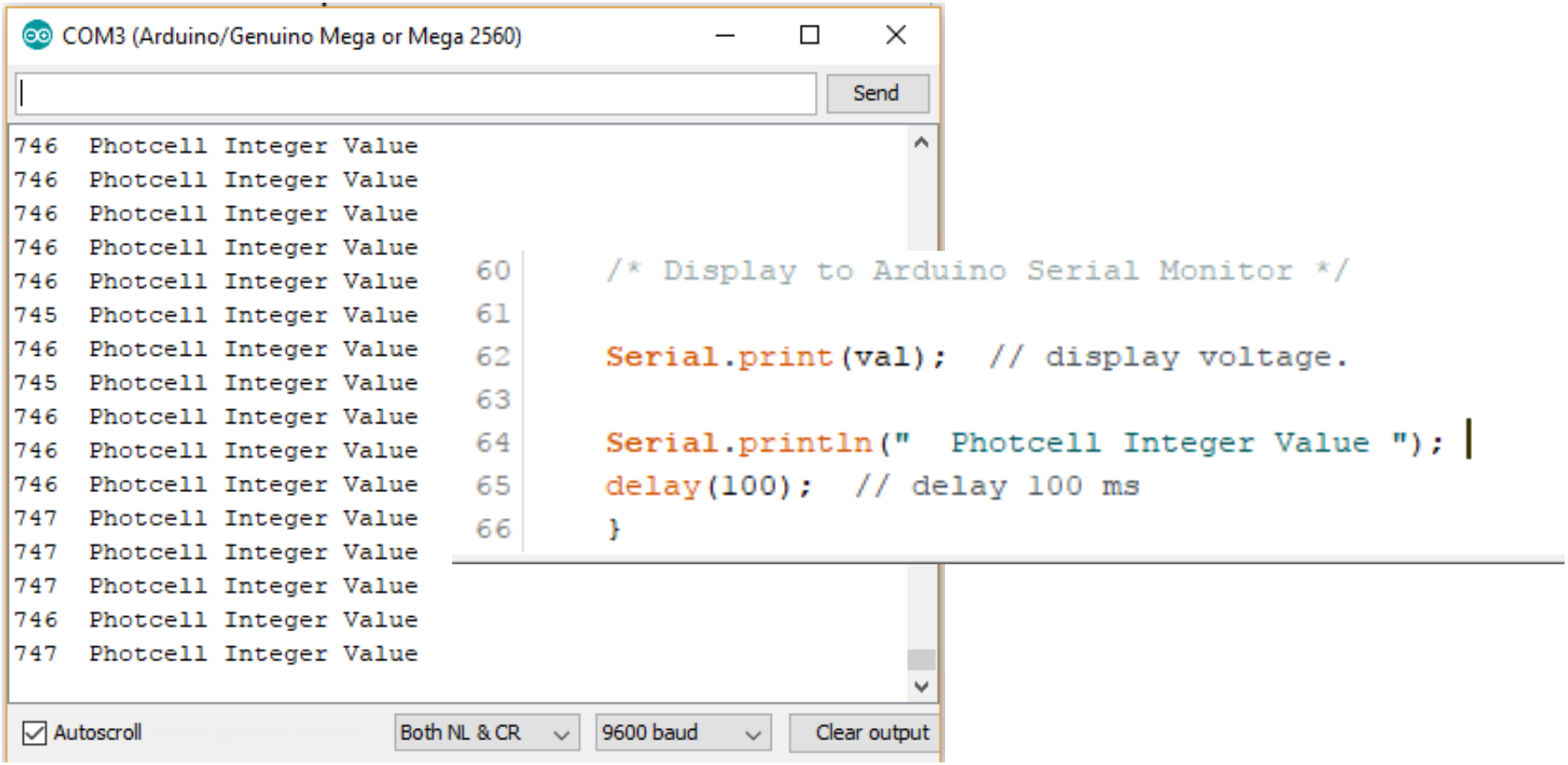
```
25      void loop()
26    {
27    int val = analogRead(A0);
28
29    if (val < 140)
30    {
31      digitalWrite(Red_Led,HIGH);
32      digitalWrite(Yel_Led,HIGH);
33      digitalWrite(Grn_Led,LOW);
34
```



Voltage increases when darker.



Voltage increases when brighter.


COM3 (Arduino/Genuino Mega or Mega 2560)
Send
746 Photcell Integer Value
746 Photcell Integer Value
746 Photcell Integer Value
746 Photcell Integer Value
746 Photcell Integer Value
745 Photcell Integer Value
746 Photcell Integer Value
745 Photcell Integer Value
746 Photcell Integer Value
746 Photcell Integer Value
746 Photcell Integer Value
747 Photcell Integer Value
747 Photcell Integer Value
747 Photcell Integer Value
746 Photcell Integer Value
747 Photcell Integer Value
Autoscroll
Both NL & CR
9600 baud
Clear output
60 /* Display to Arduino Serial Monitor */
61
62 Serial.print(val); // display voltage.
63
64 Serial.println(" Photcell Integer Value ");
65 delay(100); // delay 100 ms
66 }

Arduino Compile Errors

```
29
30      serial.print(state_set);
31
32      delay(100);
```

'serial' was not declared in this scope

What is the error?

Serial should be Uppercase.

```
29
30      Serial.print(state_set);
31
32      delay(100);
```

Arduino Compile Errors

```
19    {
20    int temperature = analogRead(a0);
21    boolean state_set = digitalRead(set_sw); // read digital inputs
```

'a0' was not declared in this scope

What is the error?

```
20    int temperature = analogRead(A0);
21    boolean state_set = digitalRead(set_sw); // read digital inputs
```

Done uploading.

What is the error?

Arduino Compile Errors

```
15
16 LiquidCrystal lcd(8, 9, 4, 5, 6, 7); / select the pins used on the LCD panel
17
```

'select' does not name a type

What is the error?

```
15
16 LiquidCrystal lcd(8, 9, 4, 5, 6, 7); // select the pins used on the LCD panel
17
```

Done uploading.

Arduino Compile Errors

```
27
28      float temperature = dht.readTemperature();
29      float humidity = dht.readHumidity();
30
31      lcd.setCursor(0,0);             // set the LCD cursor   position top line left
32      lcd.print(temperature);
33      lcd.print("  deg C");
34
35      float humidity = dht.readHumidity();
36
```

redeclaration of 'float humidity'

What is the error? Declared humidity a second time.

```
21
22      lcd.setCursor(0,1);
23      lcd.print(Lab 3");
24
```

missing terminating " character

What is the error?

"Lab 3"

```
16 void loop(){
17
18      int ButtonValue = analogRead(A0);  10 bit A/D 5.0
19
20      lcd.setCursor(0,0);        // set the LCD cursor
21      lcd.print("Michel ");
```

expected ';' before 'bit'

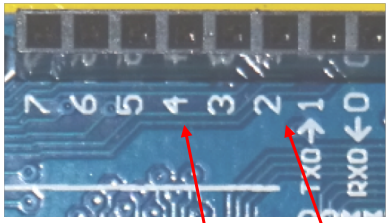
What is the error?

Missing // comment .

File Edit Sketch Tools Help

Lab_2_Part_1_Set_Reset_transistor_red_led §

```
1   /*  Lab 2 Part 1 CAM8302E
2    *  Modified by Michel Hanbury
3    *  September 28th 2015 */
4
5   #define set_sw  4
6   #define clear_sw   5 // digital inputs
7   #define RedLed  2
8
9   void setup()
10  {
11  Serial.begin(9600); // set bit rate to 9600 bps
12
13  pinMode(set_sw,INPUT);
14  pinMode(clear_sw,INPUT);
15  pinMode(RedLed, OUTPUT);  // on power up pins default to input
16
17  }
```

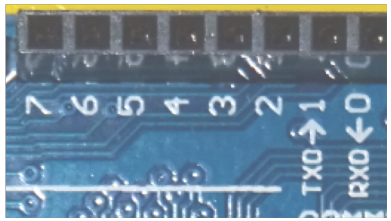


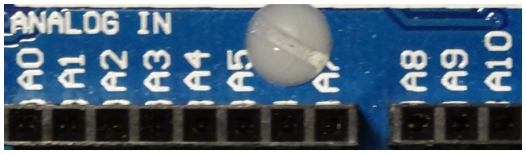
Pin 2
Digital Output

Pin 4
Digital input

Arduino I/O setup

```
18  void loop() {
19
20  int temperature = analogRead(A0); // 10 bit analog input
21  boolean state_set = digitalRead(set_sw); // read digital inputs
22  boolean state_clear =  digitalRead(clear_sw);
23
24  if (state_set == LOW)    // test set switch for closure
25    digitalWrite(RedLed, HIGH);  // output high (5.0 volts)
26
27  if (state_clear == LOW) // test clear switch for low
28     digitalWrite(RedLed, LOW);  // output low (0.0 volts)
29
30   Serial.print("temperature =  ");
31   Serial.print(temperature);
32   Serial.print(" \t ");
33   Serial.print(state_set);
34   Serial.print(" \t ");
35   Serial.println(state_clear);
36
37   delay(100);
38  }
```

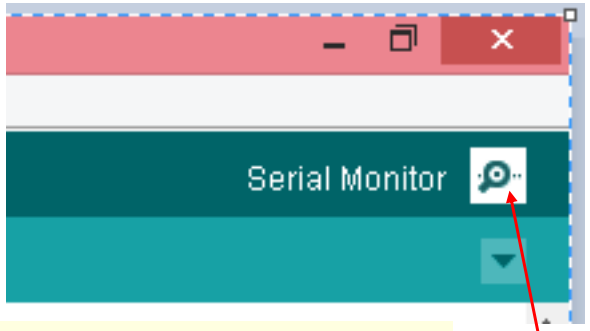




Arduino I/O setup

```
temperature =  531    0    0
temperature =  533    0    0
temperature =  532    1    0
temperature =  535    0    0
temperature =  535    0    0
temperature =  535    0    1
temperature =  536    1    0
temperature =  539    1    0
temperature =  540    0    0
temperature =  539    0    0
temperature =  540    1    0
temperature =  540    0    0
temperature =  543    0    1
temperature =  543    0    0
temperature =  543    1    0
temperature =  543    1    0
temperature =  543    0    0
temperature =  543    0    0
temperature =  543    0    0
temperature =  545    0    0
temperature =  545    1    1
temperature =  545    1    1
```

Sensor Values and input status.



Serial Port Viewer

Thermistor temperature in degC versus resistance in ohms

10,0	19691	26,0	9577,7
11,0	18788	27,0	9175,6
12,0	17932	28,0	8792,6
13,0	17120	29,0	8427,7
14,0	16350	30,0	8080,0
15,0	15618	31,0	7748,5
16,0	14923	32,0	7432,4
17,0	14236	33,0	7131,0
18,0	13636	34,0	6843,4
19,0	13040	35,0	6569,0
20,0	12474	36,0	6307,0
21,0	11928	37,0	6057,0
22,0	11409	38,0	5818,1
23,0	10915	39,0	5590,0
24,0	10446	40,0	5372,0
25,0	10000		

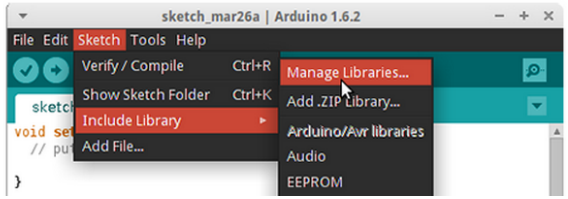
What are Libraries?

Libraries are a collection of code that makes it easy for you to connect to a sensor, display, module, etc. For example, the built-in LiquidCrystal library makes it easy to talk to character LCD displays. There are hundreds of additional libraries available on the Internet for download. The built-in libraries and some of these additional libraries are listed in the reference. To use the additional libraries, you will need to install them.

How to Install a Library

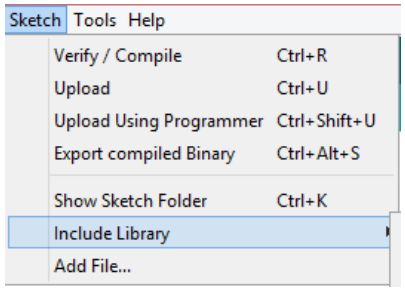
Using the Library Manager

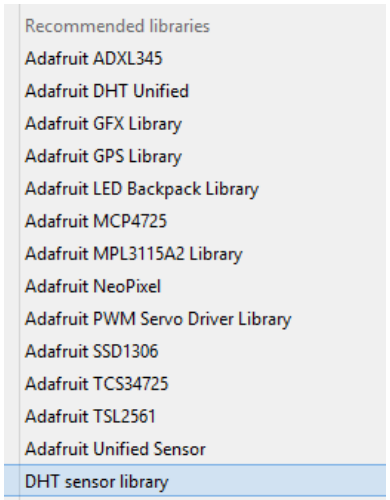
To install a new library into your Arduino IDE you can use the Library Manager (available from IDE version 1.6.2). Open the IDE and click to the "Sketch" menu and then *Include Library > Manage Libraries*.



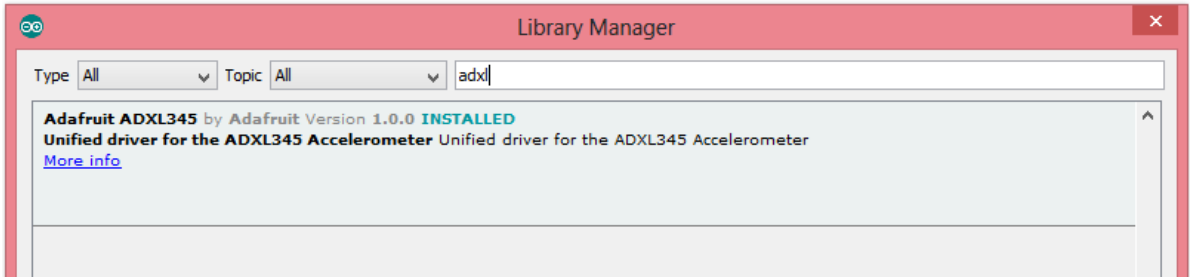
Standard Libraries

- EEPROM - reading and writing to "permanent" storage
- Ethernet - for connecting to the internet using the Arduino Ethernet Shield
- Firmata - for communicating with applications on the computer using a standard serial protocol.
- GSM - for connecting to a GSM/GRPS network with the GSM shield.
- LiquidCrystal - for controlling liquid crystal displays (LCDs)
- SD - for reading and writing SD cards
- Servo - for controlling servo motors
- SPI - for communicating with devices using the Serial Peripheral Interface (SPI) Bus
- SoftwareSerial - for serial communication on any digital pins. Version 1.0 and later of Arduino incorporate Mikal Hart's NewSoftSerial library as SoftwareSerial.
- Stepper - for controlling stepper motors
- TFT - for drawing text , images, and shapes on the Arduino TFT screen
- WiFi - for connecting to the internet using the Arduino WiFi shield
- Wire - Two Wire Interface (TWI/I2C) for sending and receiving data over a net of devices or sensors.





https://www.arduino.cc/en/Guide/Libraries

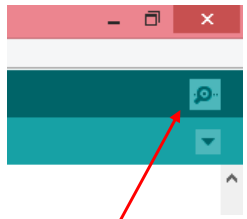


```
8  #include <DHT.h>

9

10 #define DHTPIN 22     // what pin we're connected to
11 #define DHTTYPE DHT22   // DHT 22  (AM2302)
12
13 // Initialize DHT sensor for normal 16mhz Arduino
14    DHT dht(DHTPIN, DHTTYPE);

15
16 void setup(){
17    Serial.begin(9600); // set bit rate to 9600 bps
18     }
19 void loop(){
20
21     float temperature = dht.readTemperature();
22     float humidity = dht.readHumidity();
23
24     Serial.print(temperature);
25     Serial.print("  deg C    ");
26
27     Serial.print(humidity);
28     Serial.println("  % hum");
29
30     delay(2000);               // wait 2 seconds
31     }
32
```

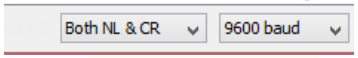
DHT Library

22,DHT22

Library Functions

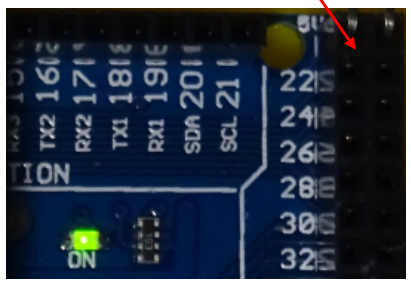


Serial Port Viewer

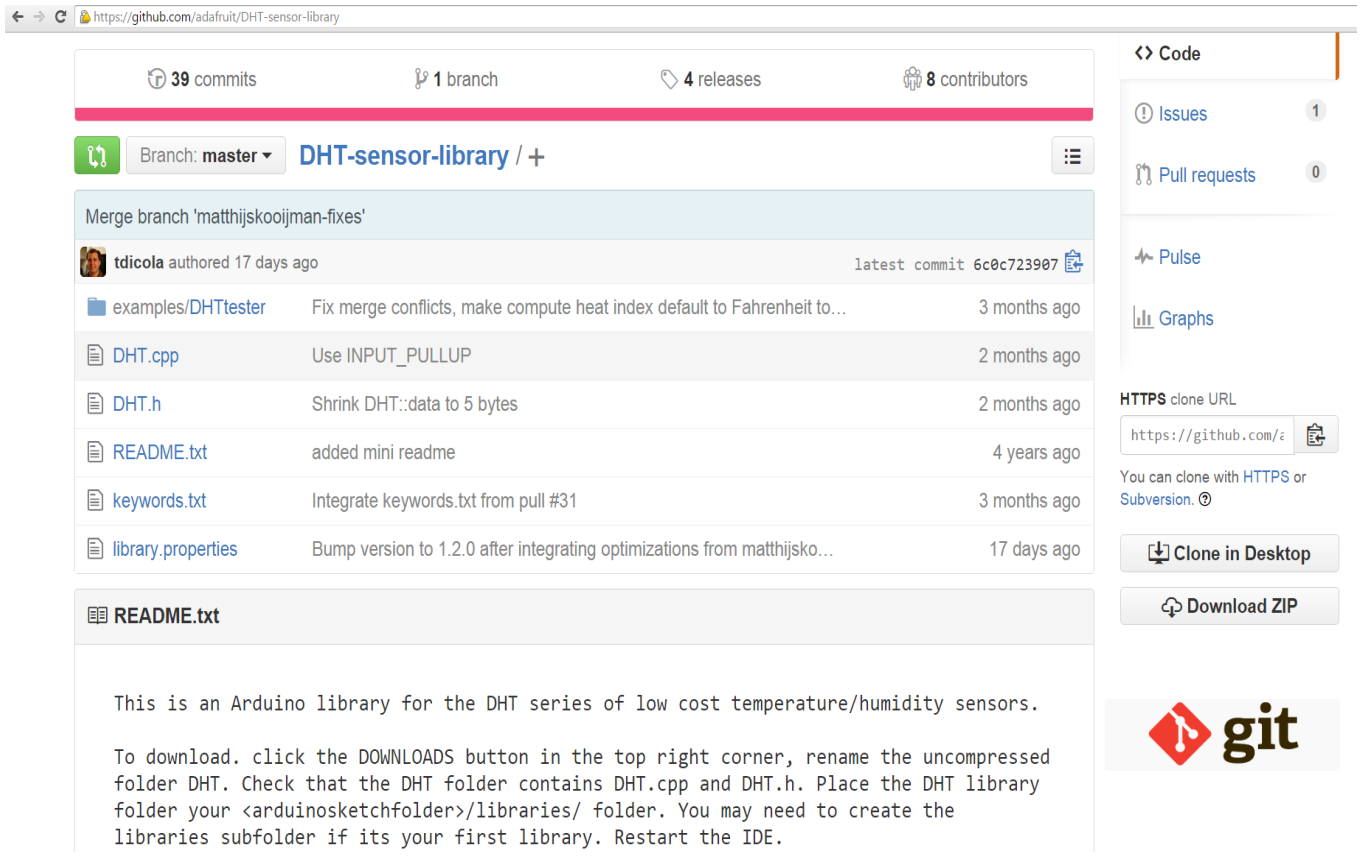


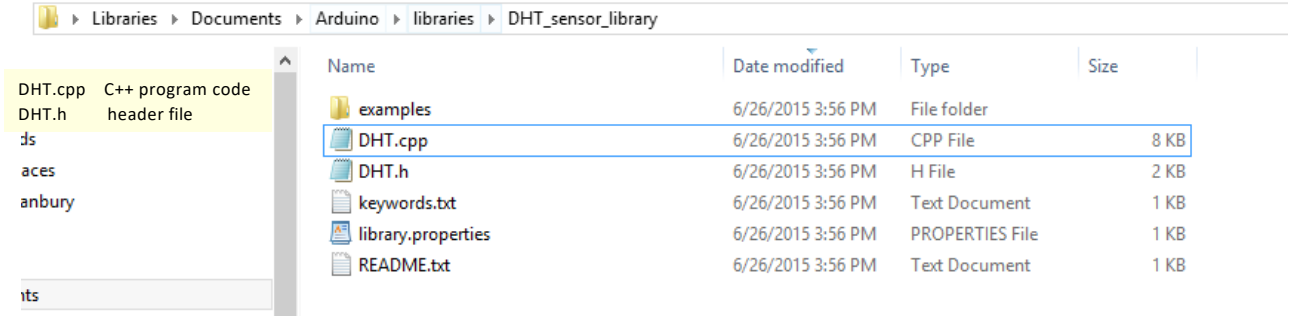
Baud must match setting in your program.

Sensor Pin (22)



https://github.com/adafruit/DHT-sensor-library






```
#include <DHT.h>

#define DHTPIN 22     // what pin we're connected to
#define DHTTYPE DHT22   // DHT 22  (AM2302)

// Initialize DHT sensor for normal 16mhz Arduino
   DHT dht(DHTPIN, DHTTYPE);

void setup(){
```

```
float temperature = dht.readTemperature();
float humidity = dht.readHumidity();
```

https://www.arduino.cc/en/Guide/Libraries

```
  // Now read the 40 bits sent by the sensor.  Each bit is sent as a 50
  // microsecond low pulse followed by a variable length high pulse.  If the
  // high pulse is ~28 microseconds then it's a 0 and if it's ~70 microseconds
  // then it's a 1.  We measure the cycle count of the initial 50us low pulse
  // and use that to compare to the cycle count of the high pulse to determine
  // if the bit is a 0 (high state cycle count < low state cycle count), or a
  // 1 (high state cycle count > low state cycle count).
  for (int i=0; i<40; ++i) {
    uint32_t lowCycles = expectPulse(LOW);
    if (lowCycles == 0) {
      DEBUG_PRINTLN(F("Timeout waiting for bit low pulse."));
      _lastresult = false;
      return _lastresult;
    }
    uint32_t highCycles = expectPulse(HIGH);
    if (highCycles == 0) {
      DEBUG_PRINTLN(F("Timeout waiting for bit high pulse."));
      _lastresult = false;
      return _lastresult;
    }
    data[i/8] <<= 1;
    // Now compare the low and high cycle times to see if the bit is a 0 or 1.
    if (highCycles > lowCycles) {
      // High cycles are greater than 50us low cycle count, must be a 1.
      data[i/8] |= 1;
    }
    // Else high cycles are less than (or equal to, a weird case) the 50us low
    // cycle count so this must be a zero.  Nothing needs to be changed in the
    // stored data.
  }
```