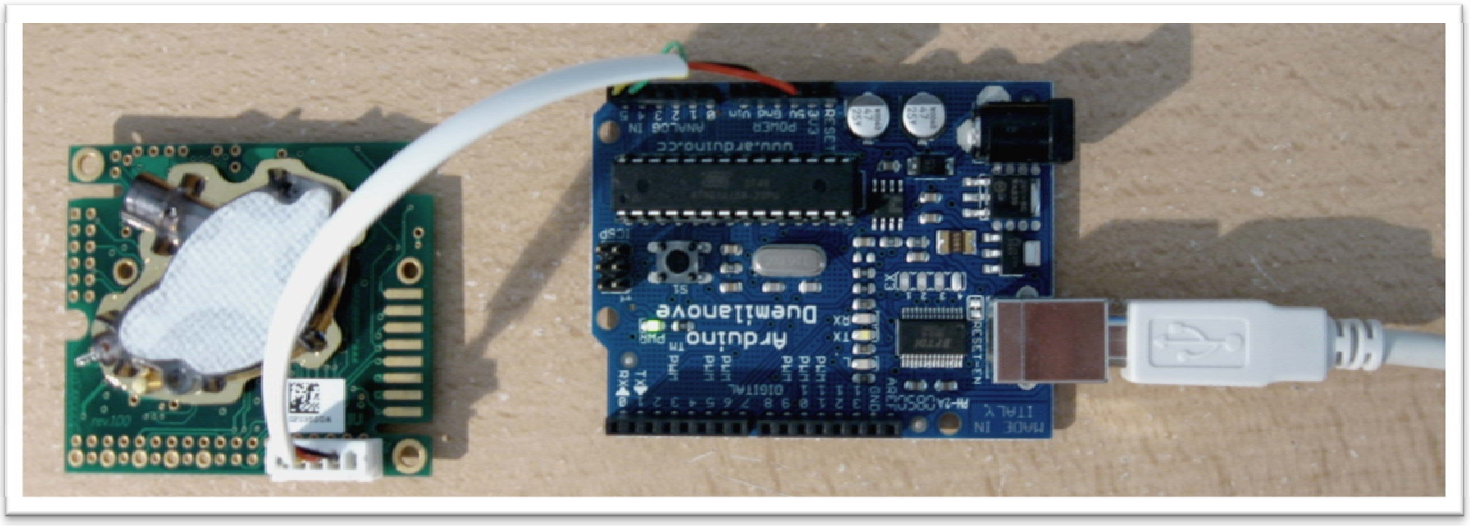
# Application Note: Interfacing with Arduino over I2C

The Arduino makes an ideal platform for prototyping and data collection with the K series of CO2 sensors.

## Electrical Connections

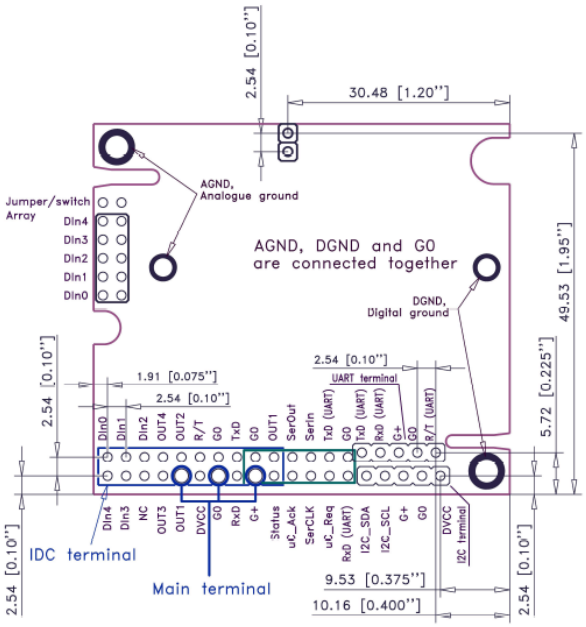


Interfacing with the sensor electrically is easy. Although the sensor runs at 3.3V it is tolerant of 5V logic levels. A direct electrical connection is possible using the Arduino's hardware I2C pins as follows:

- Arduino analog input 5 - **I2C SCL**
- Arduino analog input 4 - **I2C SDA**

Both the Arduino and CO2 sensor have built-in pull-up resistors.

The sensor will be wired using the I2C terminal in the following drawing:



# Software Interface

We will use the built in Wire library to interface with the K30 CO2 sensor. Import it into the project and initialize it in the setup() routine:

```
#include <Wire.h>

int co2Addr = 0x68;  // This is the default address of the CO2 sensor, 7bits shifted left.

void setup() {
  Serial.begin(9600);
  Wire.begin ();
  pinMode(13, OUTPUT);  // We will use this pin as a read-indicator
  Serial.println("What a wonderful day, to read atmospheric CO2 concentrations!");
}
```

Next we will start polling the sensor using the standard I2C sequences, taken from Appendix A of the I2C Com Guide, available on our website:

```
  Wire.beginTransmission(co2Addr);
  Wire.send(0x22);
  Wire.send(0x00);
  Wire.send(0x08);
  Wire.send(0x2A);
  Wire.endTransmission();

  delay(10);

  Wire.requestFrom(co2Addr, 4);

  byte i = 0;
  byte buffer[4] = {0, 0, 0, 0};

  while(Wire.available())
  {
    buffer[i] = Wire.receive();
    i++;
  }
```

From this point it is a simple matter of taking the bytes in the buffer and converting them into a CO2 value. This will be left as an exercise in the included sample code.

Additional command sequences can be found in the Com Guide however for most applications CO2 reading is all that will be necessary.

# Appendix A: Sample Code

```
// CO2 Meter K-series Example Interface
// by Andrew Robinson, CO2 Meter <co2meter.com>

// Talks via I2C to K30/K22/K33/Logger sensors and displays CO2 values
// 12/31/09

#include <Wire.h>

// We will be using the I2C hardware interface on the Arduino in
// combination with the built-in Wire library to interface.


//  Arduino analog input 5 - I2C SCL
//  Arduino analog input 4 - I2C SDA

/*
  In this example we will do a basic read of the CO2 value and checksum verification.
  For more advanced applications please see the I2C Comm guide.
*/

int co2Addr = 0x68;
// This is the default address of the CO2 sensor, 7bits shifted left.

void setup() {
  Serial.begin(9600);
  Wire.begin ();
  pinMode(13, OUTPUT);  // We will use this pin as a read-indicator
  Serial.println("What a wonderful day, to read atmospheric CO2 concentrations!");
}


////////////////////////////////////////////////////////////////
// Function : int readCO2()
// Returns  : CO2 Value upon success, 0 upon checksum failure
// Assumes  : - Wire library has been imported successfully.
//            - LED is connected to IO pin 13
//            - CO2 sensor address is defined in co2_addr
////////////////////////////////////////////////////////////////
int readCO2()
{
  int co2_value = 0;
  // We will store the CO2 value inside this variable.

  digitalWrite(13, HIGH);
  // On most Arduino platforms this pin is used as an indicator light.

  //////////////////////////
  /* Begin Write Sequence */
  //////////////////////////

  Wire.beginTransmission(co2Addr);
  Wire.send(0x22);
  Wire.send(0x00);
  Wire.send(0x08);
  Wire.send(0x2A);
```

```
Wire.endTransmission();

//////////////////////////
/* End Write Sequence. */
//////////////////////////

/*
   We wait 10ms for the sensor to process our command.
   The sensors's primary duties are to accurately
   measure CO2 values. Waiting 10ms will ensure the
   data is properly written to RAM

*/

delay(10);

//////////////////////////
/* Begin Read Sequence */
//////////////////////////

/*
   Since we requested 2 bytes from the sensor we must
   read in 4 bytes. This includes the payload, checksum,
   and command status byte.

*/

Wire.requestFrom(co2Addr, 4);

byte i = 0;
byte buffer[4] = {0, 0, 0, 0};

/*
   Wire.available() is not nessessary. Implementation is obscure but we leave
   it in here for portability and to future proof our code
*/
while(Wire.available())
{
  buffer[i] = Wire.receive();
  i++;
}

//////////////////////////
/* End Read Sequence */
//////////////////////////

/*
   Using some bitwise manipulation we will shift our buffer
   into an integer for general consumption
*/

co2_value = 0;
co2_value |= buffer[1] & 0xFF;
co2_value = co2_value << 8;
co2_value |= buffer[2] & 0xFF;


byte sum = 0;                              //Checksum Byte
sum = buffer[0] + buffer[1] + buffer[2];   //Byte addition utilizes overflow
```

```
  if(sum == buffer[3])
  {
    // Success!
    digitalWrite(13, LOW);
    return co2_value;
  }
  else
  {
    // Failure!
    /*
      Checksum failure can be due to a number of factors,
      fuzzy electrons, sensor busy, etc.
    */

    digitalWrite(13, LOW);
    return 0;
  }
}

void loop() {

    int co2Value = readCO2();
    if(co2Value > 0)
    {
      Serial.print("CO2 Value: ");
      Serial.println(co2Value);
    }
    else
    {
      Serial.println("Checksum failed / Communication failure");
    }

  delay(2000);
}
```

**Note:** Checksum failures do happen periodically. These can be partially avoided by increasing the delay time between writing the request packet and polling for a response. However they cannot be eliminated. The sensor's primary functions are related to accurately measuring CO2 concentrations and as a side effect communication is often delayed as crucial, time-sensitive operations are taken place. Plan your code accordingly. A more advanced application would involve proper retry logic.