CO2Meter.com
131 Business Center Dr. Unit A-3, Ormond Beach, FL 32174
TEL: 877.678.4259 | FAX: 866.422.2356
EMAIL: sales@co2meter.com | www.co2meter.com

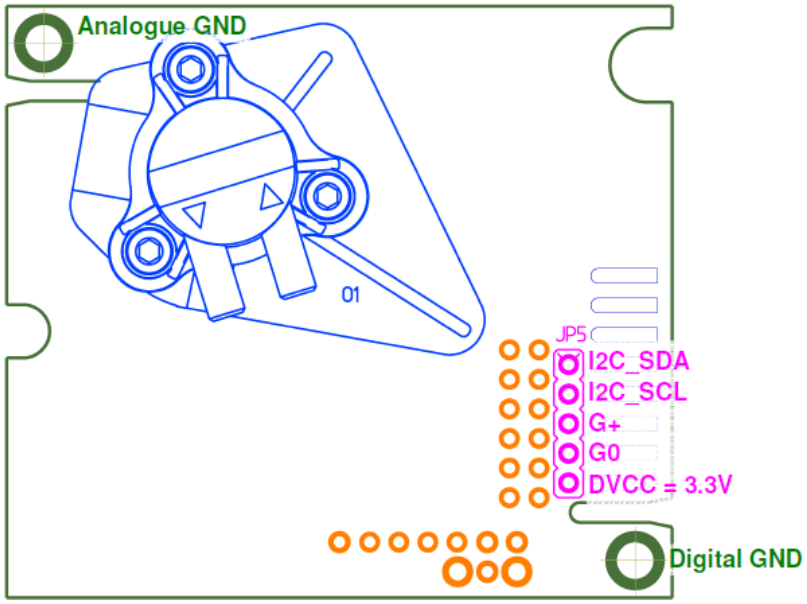
# AN113 – Using the K33 ELG/BLG Sensor with a Microcontroller

## Overview

The K33 Platform was primarily designed as a data collection platform, and as such has features designed to ensure power consumption is kept as low as possible. These feature can make interfacing it with a typical microcontroller unintuitive at first. The purpose of this document is to detail the steps required to connect a K33 ELG/BLG sensor to a microcontroller (in this situation we'll be using the Arduino platform) and initiate data collection.

## Physical Connections & Power Requirements

Due to the power requirements of the K33 platform, powering the sensor directly from the 5V typically found on the Arduino is problematic and can result in failed communication and inaccurate readings. We recommend powering your sensor with an external 6V battery pack or wall supply. Connect power to G+ and ground to G0 terminals of JP5. Connect the I2C lines to I2C_SDA and I2C_SCL on the device. Using an Arduino, the SCL line is Analog In 5, and the SDA is Analog In 4.




## Timing

### *The Wakeup Pulse*

The most significant obstacle to implementing an I2C routine to communicate to the K33 ELG/BLG sensor is to ensure that a pulse of sufficient length is sent to the device before initiating communication to wake the sensor up from its sleep state. This is done by pulsing the SDA line before initiating the I2C communication requests. In many modern microcontrollers the I2C implementation is done in hardware and this is difficult to accomplish. An acceptable solution, and the one we used here, will be to temporarily disable the I2C engine on the device, pulse the line, re-enable the I2C engine, and wait a millisecond to begin communication. In Appendix A this is implemented in wakeupSensor() for the Atmega168 microcontroller.


### *Communication During Measurement*

A second hazard to avoid is communicating with the sensor during a measurement cycle. The K33 ELG/BLG sensor takes a measurement using a series of 25 IR lamp pulses and absorption measurements. During these pulses communication with host devices is not a priority and will be delayed or dropped to maintain accuracy. A robust retry algorithm is required to compensate. Alternatively, if the host is initiating all the communication we can use this to our advantage and simply avoid communication while the sensor is taking measurements.

## Order of Events

A typical measurement cycle is broken into multiple steps:

1. Send “Initiate Measurement” Command to Sensor
2. Wait 16 seconds while device takes data
3. Read CO2, Temperature, and RH
4. Wait an additional 9 seconds to avoid internal heating affecting measurement accuracy.

## Implementation

The code in this document has been tested on an Arduino Duemilanove board running an Atmega168 microcontroller. In the wake sensor function we directly access some device registers, and additionally we make use of the I2C bus. To ensure this code will work on your chosen development platform please ensure that the I2C bus is located on the correct lines and the registers transfer over correctly.

Implementation on a different platform should be fairly straightforward, any modern hardware I2C engine will be able to successfully talk to the K33 ELG/BLG sensor, as long as the manual wake-up pulse can be provided.

131 Business Center Dr. Unit A-3, Ormond Beach, FL 32174
TEL: 877.678.4259 | FAX: 866.422.2356
EMAIL: sales@co2meter.com | www.co2meter.com

# Appendix A – Code Listing

```
// CO2 Meter K-series Example Interface
// by Andrew Robinson, CO2 Meter <co2meter.com>
// Talks via I2C to K33-ELG/BLG Sensors for Host-Initiated Data Collection
// 4.1.2011
#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(115200);
  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 : void wakeSensor()
// Executes : Sends wakeup commands to K33 sensors.
// Note     : THIS COMMAND MUST BE MODIFIED FOR THE SPECIFIC AVR YOU ARE USING
//            THE REGISTERS ARE HARD-CODED
////////////////////////////////////////////////////////////////////

void wakeSensor() {
  // This command serves as a wakeup to the CO2 sensor, for K33-ELG/BLG Sensors Only

  // You'll have the look up the registers for your specific device, but the idea here is simple:
  // 1. Disabled the I2C engine on the AVR
  // 2. Set the Data Direction register to output on the SDA line
  // 3. Toggle the line low for ~1ms to wake the micro up. Enable I2C Engine
  // 4. Wake a millisecond.

  TWCR &= ~(1<<2); // Disable I2C Engine
  DDRC |= (1<<4); // Set pin to output mode
  PORTC &= ~(1<<4); // Pull pin low
  delay(1);
  PORTC |= (1<<4); // Pull pin high again
  TWCR |= (1<<2); // I2C is now enabled
  delay(1);
}

////////////////////////////////////////////////////////////////////
// Function : void initPoll()
// Executes : Tells sensor to take a measurement.
// Notes    : A fuller implementation would read the register back and
//            ensure the flag was set, but in our case we ensure the poll
//            period is >25s and life is generally good.
////////////////////////////////////////////////////////////////////
```

131 Business Center Dr. Unit A-3, Ormond Beach, FL 32174
TEL: 877.678.4259 | FAX: 866.422.2356
EMAIL: sales@co2meter.com | www.co2meter.com

```
void initPoll() {
  Wire.beginTransmission(co2Addr);
  Wire.send(0x11);
  Wire.send(0x00);
  Wire.send(0x60);
  Wire.send(0x35);
  Wire.send(0xA6);

  Wire.endTransmission();
  delay(20);
  Wire.requestFrom(co2Addr, 2);

  byte i = 0;
  byte buffer[2] = {0, 0};

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

}

////////////////////////////////////////////////////////////////////
// Function : double readCo2()
// Returns  : The current CO2 Value, -1 if error has occured
////////////////////////////////////////////////////////////////////

double 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();
  /*
     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(20);
  ////////////////////////
  /* 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.
```

131 Business Center Dr. Unit A-3, Ormond Beach, FL 32174
TEL: 877.678.4259 | FAX: 866.422.2356
EMAIL: sales@co2meter.com | www.co2meter.com

```
  */

  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++;
  }

  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 ((double) co2_value / (double) 1);
  }
  else {
   // Failure!
   /*
        Checksum failure can be due to a number of factors,
        fuzzy electrons, sensor busy, etc.
  */

      digitalWrite(13, LOW);
      return (double) -1;
  }
}

////////////////////////////////////////////////////////////////
// Function : double readTemp()
// Returns  : The current Temperture Value, -1 if error has occured
////////////////////////////////////////////////////////////////

double readTemp() {
  int tempVal = 0;

  digitalWrite(13, HIGH);

  Wire.beginTransmission(co2Addr);
  Wire.send(0x22);
  Wire.send(0x00);
  Wire.send(0x12);
  Wire.send(0x34);

  Wire.endTransmission();

  delay(20);
```

131 Business Center Dr. Unit A-3, Ormond Beach, FL 32174
TEL: 877.678.4259 | FAX: 866.422.2356
EMAIL: sales@co2meter.com | www.co2meter.com

```
  Wire.requestFrom(co2Addr, 4);

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

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

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

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

  if(sum == buffer[3]) {
      digitalWrite(13, LOW);
      return ((double) tempVal / (double) 100);
  }
  else {
      digitalWrite(13, LOW);
      return -1;
  }

}

////////////////////////////////////////////////////////////////////
// Function : double readRh()
// Returns  : The current Rh Value, -1 if error has occured
////////////////////////////////////////////////////////////////////

double readRh() {
   int tempVal = 0;

  digitalWrite(13, HIGH);

  Wire.beginTransmission(co2Addr);
  Wire.send(0x22);
  Wire.send(0x00);
  Wire.send(0x14);
  Wire.send(0x36);

  Wire.endTransmission();

  delay(20);

  Wire.requestFrom(co2Addr, 4);

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

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

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

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

  if(sum == buffer[3]) {
      digitalWrite(13, LOW);
      return (double) tempVal / (double) 100;
  }
  else {
      digitalWrite(13, LOW);
      return -1;
  }
}

void loop() {
  // We keep the sample period >25s or so, else the sensor will start ignoring sample requests.
  wakeSensor();
  initPoll();

  delay(16000);
  wakeSensor();
  double tempValue = readTemp();

  delay(20);
  wakeSensor();
  double rhValue = readRh();

  delay(20);
  wakeSensor();
  double co2Value = readCo2();

  if(co2Value >= 0) {
        Serial.print("CO2: ");
        Serial.print(co2Value);
        Serial.print("ppm Temp: ");
        Serial.print(tempValue);
        Serial.print("C Rh: ");
        Serial.print(rhValue);
        Serial.println("%");
  }
  else {
        Serial.println("Checksum failed / Communication failure");
  }
  delay(9000);
}
```