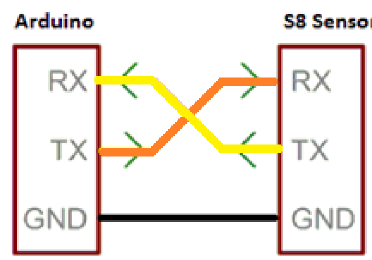
Application Note AN-162:
Connecting SenseAir's LP8 CO2 Sensor to Arduino via UART

Introduction

The Arduino Uno, Mega or Mega2560 are ideal microcontrollers for operating a LP8 sensor using a UART TXD-RXD connection. The example Ardunio code uses Software.Serial, a library built into the Arduino software.

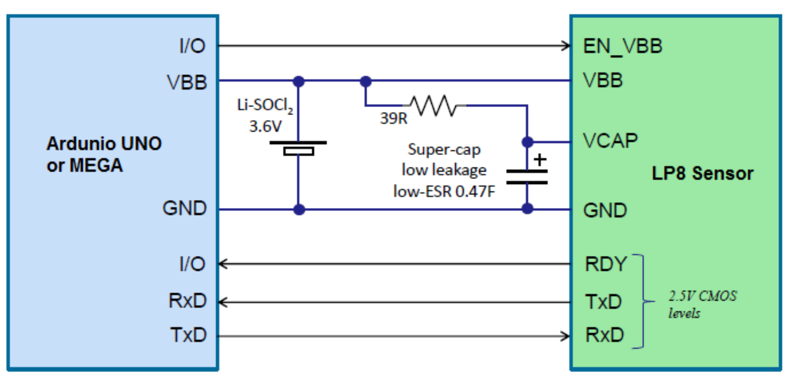
If you are new to Arduino, these low cost development boards are available from many sources. We recommend you start with authentic Arduino products. Digikey (Arduino Uno part number 1050-1041-ND) is one of many suppliers of the Arduino boards.



Serial Communications

LP8 Power Requirements

This sensor is primarily intended for battery applications, and as the diagram indicates, includes a super capacitor of 0.47 Farad. This capacitor is the primary source of power for the sensor's internal lamp drive current. The sensor can take a measurement as often as every 16 seconds; frequent measurements will decrease battery life. Below you will find detailed connection diagrams and a tested Ardunio software example.



Running the Blink Example

The best way to become familiar with the Ardunio Graphical Users Interface, or GIU, is to verify your Arduino board by running correctly. Create an Arduino project by running the example **Blink**. This simple test program confirms that a number of connection details and that the GUI are working properly.

Caution: Do not connect your Arduino board to your computer's USB port until the Arduino software is installed. Otherwise Windows would install a generic driver, not desired here.

Step 1: Install Arduino software on your computer from here:

https://www.arduino.cc/en/Main/Software. Click on **Windows Installer.**

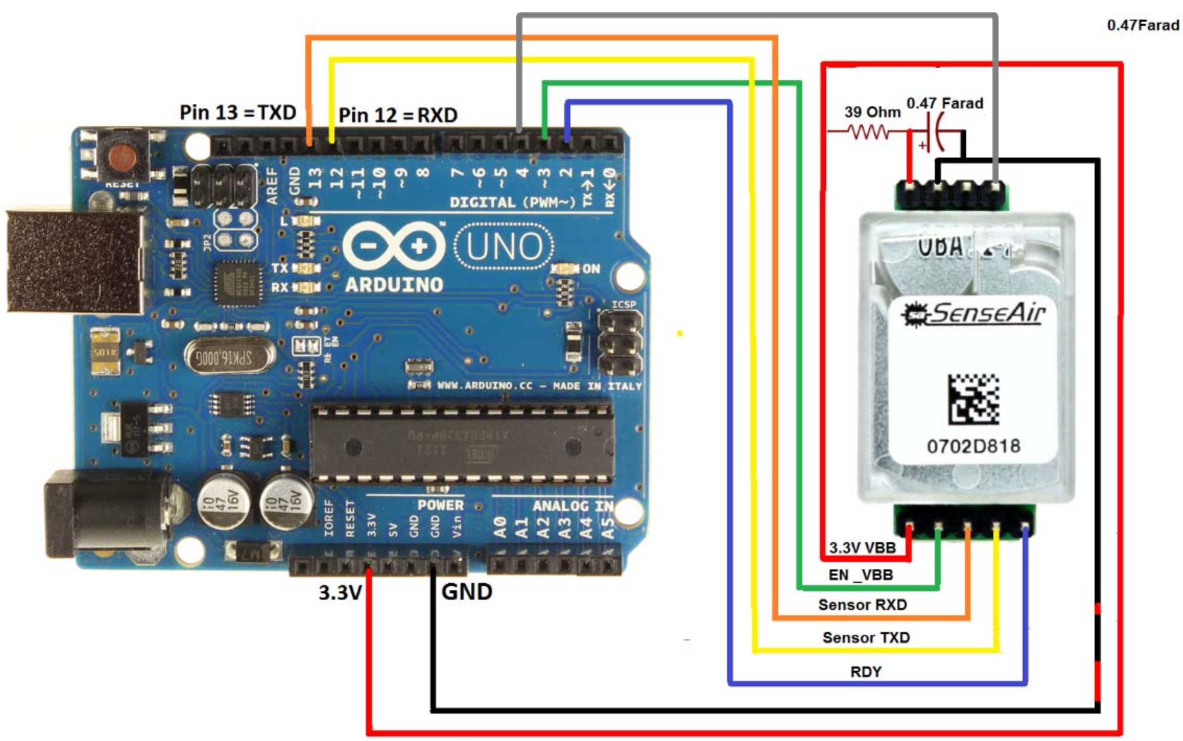
Step 2: To run Blink example, follow the instructions here:

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

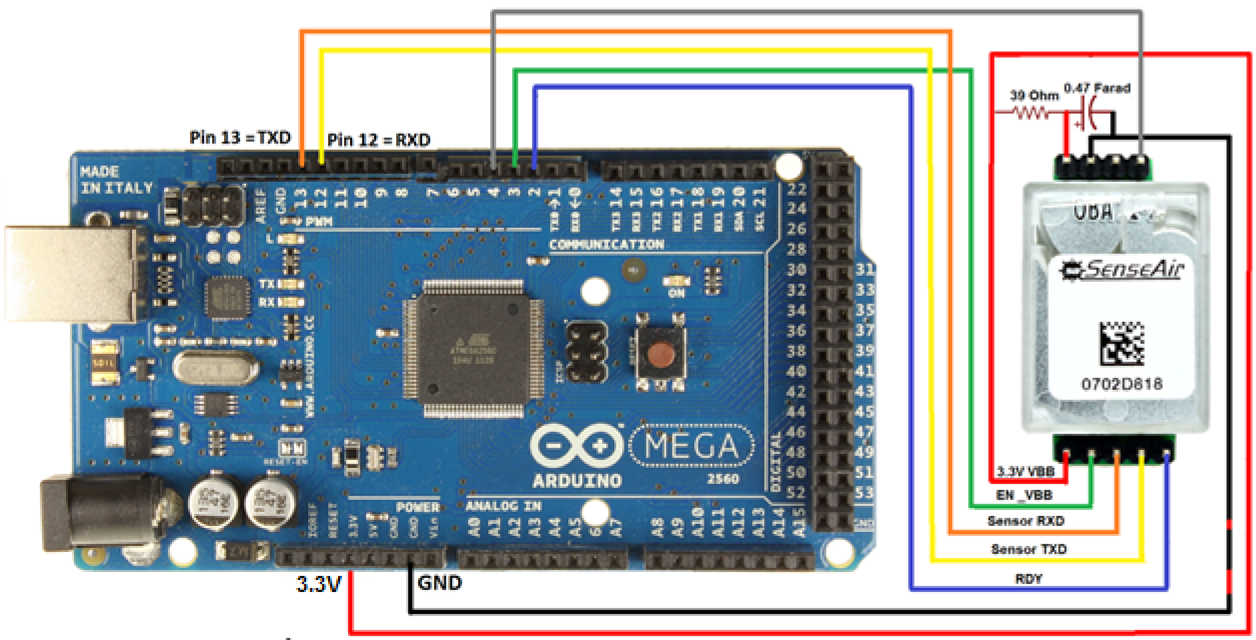
Note that Arduino MEGA has an LED on board for testing. Some Ardunio UNO versions require a LED and resistor be added as specified below.

Connecting your Arduino

Refer to the wiring diagram below for the **Arduino UNO** connecting to the LP8.



Connections to the Ardunio MEGA or MEGA 256 are identical.



Modbus Commands

The Software.serial library is used to communicate to the LP8 at 9600 BAUD. The sensor responds to ModBUS commands, a sequence of bytes, as follows:

1. First byte in the ModBUS sequence is a 0xFE.
2. Second byte is a read command, 0x44, or write command, 0x41.
3. Third byte is Sensor RAM address High byte.
4. Fourth byte is Sensor RAM address Low byte.
5. Fifth byte is number of bytes to be transmitted to ore received by the LP8.
6. Sixth and 7^{th} bytes are CRC low and CRC High bytes.

The Ardunio code (sketch) below includes a function that communicates to or from the sensor array:

Send Request Function

The function is defined as:

```
sendRequest(function_name, m, n);
```

where `function_name` contains the actual ModBUS command bytes,
`m` is the number of bytes in the Modbus command, and
`n` is the number of bytes returned from the sensor.

ModBUS CRC Calculation

The function call is defined as:

```
ModRTU_CRC(byte sensor_data[], byte n1)
```

where `byte sensor_data[]` is an Ardunio RAM array,and

`bytes n1` is the number of bytes, returning a 16-bit result.

Arduino Control of LP8 Sensor

Since the LP8 is a low power sensor, the LP8 can be turned off between measurements.

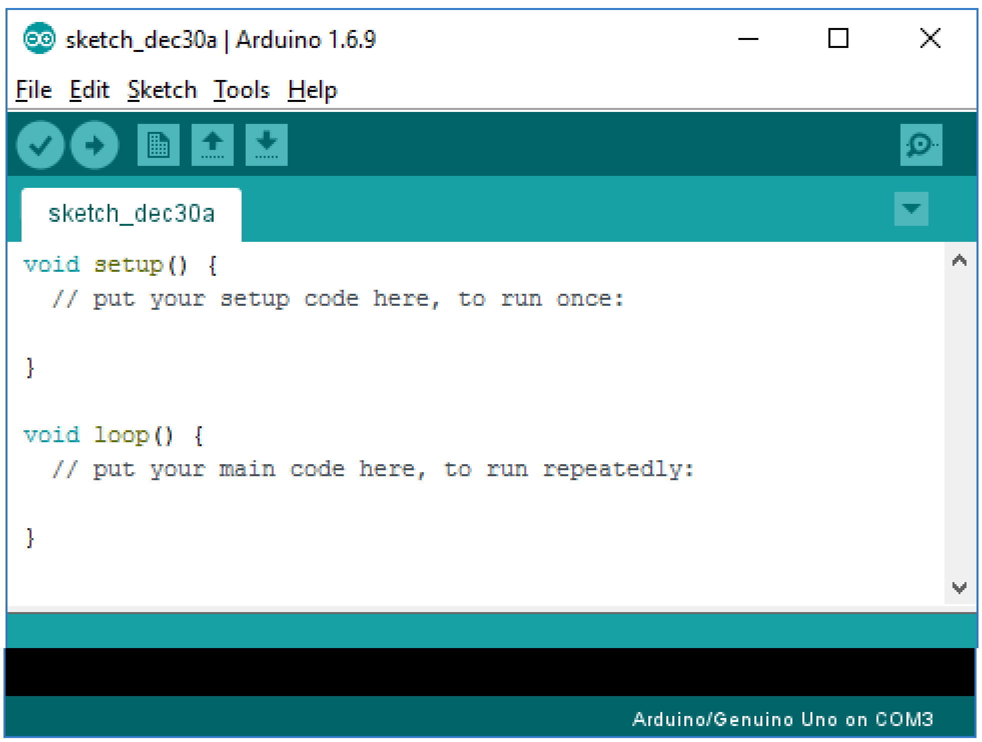
The first time the LP8 and Ardunio are connected to a 3.6 V battery, the following steps are required:

1. Reset device:
   a. reset_device( ) powers up the LP8 via VBB_Enable, and monitors RDY pin will go low for approx. 300 msec, then to go high.
2. Initialize the LP8: sendRequest(simple_write, 8, 4); // send to address 0x0080 a 0x10
   a. The ModBUS command for simple_write[] = {0xfe, 0x41, 0x00, 0x80, 0x01, 0x10, 0x28, 0x7e}// CRC bytes = 0x28, 0x7e
3. Copy 32 bytes of LP8 RAM into Ardunio RAM array response[].
   a. sendRequest(read_32_bytes, 7, 37); //37 refers to the total number of bytes read from the LP8.
   b. read_32_bytes[] = {0xfe, 0x44, 0x00, 0x80, 0x20, 0x79, 0x3C}; // CRC = 0x79, 0x3C
4. The CO2 concentration in PPM is located in the response[29] and response[30].
5. Shut down the LP8 by setting EN_Vbb to 0 V.

6. For subsequent measurements do the following:
    a. Set EN_Vbb to 3-3.3V which powers up the sensor.
    b. Wait ~300 mec for RDY to go high.
    c. Generate a transmit-packet based on the Ardunio RAM data plus a newly calculated 16-bit CRC.
    d. Send this packet to the LP8, Addresses 0x0081 thru 0x0099.
    e. Read sensor RAM locations 0x0081 0x0099 and save in Ardunio RAM.
    f. CO2 results are available at response[29] and response[30].
    g. Turn off the sensor by setting EN_Vbb to 0 V.
    h. For your next measurement, go to step 6 above.

Creating an Arduino to LP8 Project

1. Open the Ardunio GUI and select File > New



2. Select all the text in sketch and delete it.
3. Copy Appendix A Ardunio code (below) into the GUI area that you just deleted.
4. Select Sketch > Verify/Compile.
5. Select Sketch > Upload
6. Select Tools > Serial Monitor. The program outputs its activity using Serial.print statements.
7. Observe the Output as shown in Appendix B.

Appendix A: Sample Source Code

```
// 12/30/16 LP8 correctly creates transmit packet+ CRC, sends and
turns sensor power off, then on
#include <SoftwareSerial.h>
/*
  Basic Arduino example for LP8-Series sensor
  Created by Marv Kausch
  Co2meter.com
*/
#include "SoftwareSerial.h"
int rdy = 2;
int vbb_en = 3;
int nreset = 4;
int first_loop = 0;
int temp_cntr = 0;
SoftwareSerial LP8_Serial(12, 13); //Sets up a virtual serial port
//Using pin 12 for Rx and pin 13 for Tx

static byte simple_write[] = {0xfe, 0x41, 0x00, 0x80, 0x01, 0x10,
0x28, 0x7e}; //Write 0x010 to address 0x0080
static byte write_to_0x20[] = {0xfe, 0x41, 0x00, 0x80, 0x01, 0x20,
0x28, 0x6a};
static byte read_32_bytes[] = {0xfe, 0x44, 0x00, 0x80, 0x20, 0x79,
0x3C}; // Actual butes sent:  fe 41 00 80 01 20 28 6a
byte response[40];// No need to specifiy each entry
byte transmit_packet[45];  //Used to create a transmit packet
int crc_result = 0;

void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600); //Opens the main serial port to communicate
with the computer
  LP8_Serial.begin(9600); //Opens the virtual serial port with a
baud of 9600
  delay(1000); // one second delay
  Serial.println("\n                  Demo of
AN162_ardunio_LP8_uart");

  pinMode(rdy, INPUT);  //  Define rdy pin 2 as input
  pinMode(vbb_en, OUTPUT);  //  Define vbb_en pin 3 as output
```

```
  pinMode(nreset, OUTPUT);  //  Define nreset (!reset)pin 4 as
output
  Serial.println("Pins defined");
  digitalWrite(vbb_en, HIGH); // turn on vbb
  Serial.println("vbb ON");

}
void loop()
{

//  reset_device();  // includes a 500 msec delay
  if (first_loop == 0) {
  reset_device();  // includes a 500 msec delay
    sendRequest(simple_write, 8, 4); // send to address 0x0080 a
0x10
    first_loop = 1;
  }
  else   sendRequest(write_to_0x20, 8, 4); // send to address
0x0080 a 0x10
  delay(2000);
  unsigned long valCO2 = getValue(response);
  delay(3000);
  Serial.println("Now reading 32 bytes");
  //sendRequest(read_16_bytes,7,21);
  sendRequest(read_32_bytes, 7, 37);
  Serial.print("CO2 from 0x9a-0x9b = ");
  Serial.print(response[29], HEX);
  Serial.print(response[30], HEX);
  Serial.print(" = ");
  int decimal = 256 * response[29] + response[30];
  Serial.print(decimal);
  Serial.println("d");

  //    Call ModBUS CRC clculator

  crc_result = ModRTU_CRC(response, 35); // Function call to
calculate CRC
  Serial.print("****  CRC result =");

  int  crc_result_h = crc_result & 0xff;
  Serial.print(crc_result_h, HEX);
  Serial.print("   ");
  int crc_result_l = (crc_result >> 8) & 0xff;
  Serial.println(crc_result_l, HEX);
```

```
  // Verify that the sensor's CRC matches the just now calculated
CRC

  if ((response[35] != crc_result_h) && (response[36] ==
crc_result_l)) {
    Serial.println("! ! !  CRC Sensor and Calulated Do NOT
MATCH");
  }
  else {
    Serial.println("          CRC Sensor and Calulated MATCH");

    ////////POWER DOWN sensor here for 20 seconds
      digitalWrite(vbb_en, LOW); // turn on vbb
  Serial.println("vbb OFF");
  delay(20000);// wait 20 seconds befoe next sensor read
        digitalWrite(vbb_en, HIGH); // turn on vbb
  Serial.println("vbb ON");
    while (digitalRead(rdy) == HIGH) {
    Serial.println("waiting for rdy to go low");
  }
  Serial.println("rdy now LOW, an active LOW signal.  Low means
READY.");

    /* Since calculated and sensor CRC match, write contents of
sensor[],
        replacing received the two CRC bytes with newly calculated
CRC.
        This is necessary since a Modbus write command requires an
appropriateModBUS write command
        PLUS sensor[] data PLUS CEC added as the last 2 bytes.
        Write command is:  0xfe 0x41 0x00 0x81 0x

        Prepare transmit buffer to write Modbus commwnd PLUS
0x0081-0x099 = 0x18+1 = 0x19 = 25d
    */
    transmit_packet[0] = 0xfe; //
    transmit_packet[1] = 0x41; //Write to LP8 ModBus command
    transmit_packet[2] = 0x0; // Address H
    transmit_packet[3] = 0x80; //address L
    transmit_packet[4] = 0x19;//number of bytes = 0x0080 thru
0x0099
    transmit_packet[5] = 0x20; //INCLUDING calculation control
byte
```

```
//    for (int i = 6; i < (6+25); i++) {
    for (int i = 6; i < (0X19+6); i++) {
      transmit_packet[i] = response[i-3];
    }

 // Now compute new CRC of all data in transmit_packet[]:
   crc_result = ModRTU_CRC(transmit_packet, (6+23)); // Function
call to calculate CRC  37=0 to 36
 // Sometimes thif function responds with FFFFxxxx rather tha
xxxx.  So, strip off any possible FFFF...
  int  crc_result_h = crc_result & 0xff;
  Serial.print("transmit_packet crc_result =");
  Serial.print(crc_result_h, HEX);
  Serial.print("   ");
  int crc_result_l = (crc_result >> 8) & 0xff;
  Serial.println(crc_result_l, HEX);
  transmit_packet[37]=crc_result_h; // Add CRC to last two bytes
of transmit_packet
  transmit_packet[38]=crc_result_l;

  for (int i = 0; i < (6+25); i++) {
      Serial.print(" response[");
      Serial.print(i);
      Serial.print("] = 0x");
      Serial.print(response[i], HEX);
      Serial.print("    transmit_packet[");
      Serial.print(i);
      Serial.print("] = 0x");
      Serial.print(transmit_packet[i], HEX);
      if (i>5){
        Serial.print("  = RAM location[");
        Serial.print((0x80+i-5),HEX);
        Serial.println("]");
      }
      else Serial.print("\n");
      }

 // Send transmit_packet[] to LP8
 sendRequest(transmit_packet, 39, 4); // send 38 bytes total,
expect 4 byte response
Serial.println("\n ****** transmit_packet sent to LP8 RAM
******.");
Serial.println("Sensor response:");
temp_cntr++;
```

```
if (temp_cntr==2)while(1);// limits times thru loop for Serial
Monitor display. No other limitations
  }

}  //Bottom of main loop


void sendRequest(byte packet[], int m, int n)
{
  while (!LP8_Serial.available()) //keep sending request until we
start to get a response
  {
    Serial.println("waiting for Software.serial port
availability");
    LP8_Serial.write(packet, m);
    delay(1000); // Necessary to get consistent loading of
response[i]
  }
  int timeout = 0; //set a timeout counter
  while (LP8_Serial.available() < n ) //Wait to get a n byte
response
  {
    timeout++;
    if (timeout > 10) //if it takes too long there was probably an
error
    {
      while (LP8_Serial.available()) //flush whatever we have
        LP8_Serial.read();
      break; //exit and try again
    }
    delay(50);
  }
  for (int i = 0; i < n; i++)
  {
    response[i] = LP8_Serial.read();
    //    Serial.print("response[i] = ");
        Serial.print("response[");
        Serial.print(i);
        Serial.print("] = ");
        Serial.println(response[i], HEX);
      }
  Serial.print("\n\n");
  Serial.flush();
}
int reset_device() {
```

```
  digitalWrite(nreset, LOW); // assert RESET
  Serial.println("nreset now low");
  delay(500);//500 msec delay
  digitalWrite(nreset, HIGH); // assert RESET
  Serial.println("nreset now high");
  while (digitalRead(rdy) == HIGH) {
    Serial.println("waiting for rdy to go low");
  }
  Serial.println("rdy now LOW, an active LOW signal.  Low means 
READY.");
}

unsigned long getValue(byte packet[])
{
  int high = packet[3]; //high byte for value is 4th byte in 
packet in the packet
  int low = packet[4]; //low byte for value is 5th byte in the 
packet
  unsigned long val = high * 256 + low; //Combine high byte and 
low byte with this formula to get value
  return val;
}

/////////////////////  CRC routine here  ///////////////////

// Compute the MODBUS RTU CRC
int  ModRTU_CRC(byte sensor_data[], byte n1) {
  int ij;
  //Calc the raw_msg_data_byte CRC code
  uint16_t crc = 0xFFFF;
  String crc_string = "";
  // for (int pos = 0; pos < raw_msg_data.length()/2; pos++) {
  for (int pos = 0; pos < n1; pos++) {
    crc ^= (uint16_t)sensor_data[pos];          // XOR byte into 
least sig. byte of crc
    for (int ij = 8; ij != 0; ij--) {    // Loop over each bit
      if ((crc & 0x0001) != 0) {      // If the LSB is set
        crc >>= 1;                    // Shift right and XOR 
0xA001
        crc ^= 0xA001;
      }
      else                            // Else LSB is not set
        crc >>= 1;                    // Just shift right
    }
  }
```

```
  Serial.print("\n");  // Note, this number has low and high bytes
swapped, so use it accordingly (or swap bytes)

  Serial.print("  CRC HEX VALUE = ");
  Serial.println(crc, HEX);
  crc_string = String(crc, HEX);
  return crc;
}
```

Appendix B: Serial Monitor Sample Output

```
                  Demo of AN162_ardunio_LP8_uart
Pins defined
vbb ON
nreset now low
nreset now high
rdy now LOW, an active LOW signal.  Low means READY.
waiting for Software.serial port availability
waiting for Software.serial port availability
response[0] = FE
response[1] = 41
response[2] = 81
response[3] = E0


Now reading 32 bytes
waiting for Software.serial port availability
response[0] = FE
response[1] = 44
response[2] = 20
response[3] = 0
response[4] = 0
response[5] = 2
response[6] = 4
response[7] = 0
response[8] = 0
response[9] = 0
response[10] = 0
response[11] = 0
response[12] = 0
response[13] = 0
response[14] = 0
response[15] = 7F
response[16] = FF
response[17] = 0
response[18] = 0
response[19] = 91
response[20] = 82
response[21] = FF
response[22] = B
response[23] = 54
response[24] = 0
response[25] = 27
response[26] = 8C
response[27] = 27
response[28] = 8C
response[29] = 2
```

```
response[30] = FB
response[31] = 2
response[32] = FB
response[33] = 8
response[34] = CF
response[35] = 89
response[36] = 8D


CO2 from 0x9a-0x9b = 2FB = 763d

  CRC HEX VALUE = 8D89
****  CRC result =89   8D
          CRC Sensor and Calulated MATCH
vbb OFF
vbb ON
waiting for rdy to go low
waiting for rdy to go low
waiting for rdy to go low
waiting for rdy to go low
waiting for rdy to go low
waiting for rdy to go low
rdy now LOW, an active LOW signal.  Low means READY.

  CRC HEX VALUE = 60BB
transmit_packet crc_result =BB   60
 response[0] = 0xFE    transmit_packet[0] = 0xFE
 response[1] = 0x44    transmit_packet[1] = 0x41
 response[2] = 0x20    transmit_packet[2] = 0x0
 response[3] = 0x0    transmit_packet[3] = 0x80
 response[4] = 0x0    transmit_packet[4] = 0x19
 response[5] = 0x2    transmit_packet[5] = 0x20
 response[6] = 0x4    transmit_packet[6] = 0x0  = RAM location[81]
 response[7] = 0x0    transmit_packet[7] = 0x0  = RAM location[82]
 response[8] = 0x0    transmit_packet[8] = 0x2  = RAM location[83]
 response[9] = 0x0    transmit_packet[9] = 0x4  = RAM location[84]
 response[10] = 0x0    transmit_packet[10] = 0x0  = RAM location[85]
 response[11] = 0x0    transmit_packet[11] = 0x0  = RAM location[86]
 response[12] = 0x0    transmit_packet[12] = 0x0  = RAM location[87]
 response[13] = 0x0    transmit_packet[13] = 0x0  = RAM location[88]
 response[14] = 0x0    transmit_packet[14] = 0x0  = RAM location[89]
 response[15] = 0x7F    transmit_packet[15] = 0x0  = RAM location[8A]
 response[16] = 0xFF    transmit_packet[16] = 0x0  = RAM location[8B]
 response[17] = 0x0    transmit_packet[17] = 0x0  = RAM location[8C]
 response[18] = 0x0    transmit_packet[18] = 0x7F  = RAM location[8D]
 response[19] = 0x91    transmit_packet[19] = 0xFF  = RAM location[8E]
 response[20] = 0x82    transmit_packet[20] = 0x0  = RAM location[8F]
 response[21] = 0xFF    transmit_packet[21] = 0x0  = RAM location[90]
 response[22] = 0xB    transmit_packet[22] = 0x91  = RAM location[91]
```

```
 response[23] = 0x54    transmit_packet[23] = 0x82  = RAM location[92]
 response[24] = 0x0    transmit_packet[24] = 0xFF  = RAM location[93]
 response[25] = 0x27    transmit_packet[25] = 0xB  = RAM location[94]
 response[26] = 0x8C    transmit_packet[26] = 0x54  = RAM location[95]
 response[27] = 0x27    transmit_packet[27] = 0x0  = RAM location[96]
 response[28] = 0x8C    transmit_packet[28] = 0x27  = RAM location[97]
 response[29] = 0x2    transmit_packet[29] = 0x8C  = RAM location[98]
 response[30] = 0xFB    transmit_packet[30] = 0x27  = RAM location[99]
response[0] = FF
response[1] = FF
response[2] = FF
response[3] = FF



 ****** transmit_packet sent to LP8 RAM ******.
Sensor response:
waiting for Software.serial port availability
response[0] = FE
response[1] = 41
response[2] = 81
response[3] = E0


Now reading 32 bytes
waiting for Software.serial port availability
response[0] = FE
response[1] = 44
response[2] = 20
response[3] = 0
response[4] = 0
response[5] = 0
response[6] = 0
response[7] = 0
response[8] = 0
response[9] = 0
response[10] = 0
response[11] = 0
response[12] = 0
response[13] = 0
response[14] = 0
response[15] = 0
response[16] = 0
response[17] = 0
response[18] = 0
response[19] = 91
response[20] = 6C
response[21] = FF
response[22] = 9
```

```
response[23] = 56
response[24] = 0
response[25] = 27
response[26] = 8C
response[27] = 27
response[28] = 8C
response[29] = 3
response[30] = 2
response[31] = 3
response[32] = 2
response[33] = 8
response[34] = CD
response[35] = ED
response[36] = EF


CO2 from 0x9a-0x9b = 32 = 770d

  CRC HEX VALUE = EFED
****  CRC result =ED   EF
          CRC Sensor and Calulated MATCH
vbb OFF
vbb ON
waiting for rdy to go low
waiting for rdy to go low
waiting for rdy to go low
waiting for rdy to go low
waiting for rdy to go low
waiting for rdy to go low
rdy now LOW, an active LOW signal.  Low means READY.

  CRC HEX VALUE = 1671
transmit_packet crc_result =71   16
 response[0] = 0xFE    transmit_packet[0] = 0xFE
 response[1] = 0x44    transmit_packet[1] = 0x41
 response[2] = 0x20    transmit_packet[2] = 0x0
 response[3] = 0x0    transmit_packet[3] = 0x80
 response[4] = 0x0    transmit_packet[4] = 0x19
 response[5] = 0x0    transmit_packet[5] = 0x20
 response[6] = 0x0    transmit_packet[6] = 0x0  = RAM location[81]
 response[7] = 0x0    transmit_packet[7] = 0x0  = RAM location[82]
 response[8] = 0x0    transmit_packet[8] = 0x0  = RAM location[83]
 response[9] = 0x0    transmit_packet[9] = 0x0  = RAM location[84]
 response[10] = 0x0    transmit_packet[10] = 0x0  = RAM location[85]
 response[11] = 0x0    transmit_packet[11] = 0x0  = RAM location[86]
 response[12] = 0x0    transmit_packet[12] = 0x0  = RAM location[87]
 response[13] = 0x0    transmit_packet[13] = 0x0  = RAM location[88]
 response[14] = 0x0    transmit_packet[14] = 0x0  = RAM location[89]
 response[15] = 0x0    transmit_packet[15] = 0x0  = RAM location[8A]
```

```
 response[16] = 0x0    transmit_packet[16] = 0x0  = RAM location[8B]
 response[17] = 0x0    transmit_packet[17] = 0x0  = RAM location[8C]
 response[18] = 0x0    transmit_packet[18] = 0x0  = RAM location[8D]
 response[19] = 0x91    transmit_packet[19] = 0x0  = RAM location[8E]
 response[20] = 0x6C    transmit_packet[20] = 0x0  = RAM location[8F]
 response[21] = 0xFF    transmit_packet[21] = 0x0  = RAM location[90]
 response[22] = 0x9    transmit_packet[22] = 0x91  = RAM location[91]
 response[23] = 0x56    transmit_packet[23] = 0x6C  = RAM location[92]
 response[24] = 0x0    transmit_packet[24] = 0xFF  = RAM location[93]
 response[25] = 0x27    transmit_packet[25] = 0x9  = RAM location[94]
 response[26] = 0x8C    transmit_packet[26] = 0x56  = RAM location[95]
 response[27] = 0x27    transmit_packet[27] = 0x0  = RAM location[96]
 response[28] = 0x8C    transmit_packet[28] = 0x27  = RAM location[97]
 response[29] = 0x3    transmit_packet[29] = 0x8C  = RAM location[98]
 response[30] = 0x2    transmit_packet[30] = 0x27  = RAM location[99]
response[0] = FF
response[1] = FF
response[2] = FF
response[3] = FF


 ****** transmit_packet sent to LP8 RAM ******.
Sensor response:
```


131 Business Center Drive
Ormond Beach, FL 32174
Support (386) 256-4910
Sales (877) 678-4259
www.CO2Meter.com | Sales@CO2Meter.com