SEN-000004

MQ-2 Gas Sensor Module Lite

Description

This sensor module utilizes an MQ-2 as the sensitive component and has a protection resistor and an adjustable resistor on board. The MQ-2 gas sensor is sensitive to LPG, i-butane, propane, methane, alcohol, Hydrogen and smoke. It could be used in gas leakage detecting equipments in family and industry. The resistance of the sensitive component changes as the concentration of the target gas changes.

Features

- Continuous Analog output
- 3-pin interlock connector
- Low cost and compact size

Specifications

A. Standard Working Condition

Symbol	Parameter Name	Technical Condition	Remarks
V_C	Circuit voltage	5V±0.1	AC or DC
V_H	Heating voltage	5V±0.1	AC or DC
R_L	Load resistance	adjustable	
R_H	Heater resistance	33Kohm±5%	Room temperature
P_H	Heating consumption	Less than 800mW	

B. Environment Condition

Symbol	Parameter Name	Technical Condition	Remarks
Tao	Operating Temp.	-20°C -50°C	
Tas	Storage Temp.	-20°C -70°C	
RH	Relative Humidity	< 95%	
O_2	Oxygen Concentration	21%(standard condition) Oxygen concentration can affect sensitivity	Minimum value is 2%

C. Sensitivity Characteristics

Symbol	Parameter Name	Technical Condition	Remarks
R_S	Sensor Resistance	3Kohm-30Kohm (1000ppm iso-butane)	Detecting concentration Scope: 200ppm-5000ppm LPG and propane 300ppm-5000ppm butane 5000ppm-20000ppm methane 300ppm-5000ppm H2 100ppm-2000ppm Alcohol
α (3000ppm/1000ppm iso-butane)	Concentration slope rate	≤0.6	
Standard detecting Condition	Temp.: 20°C±2°C V_C: 5V±0.1 Humidity:65%±5% V_H:5V±0.1		
Preheating Time	Over 24 hours		

Typical Application Schematics

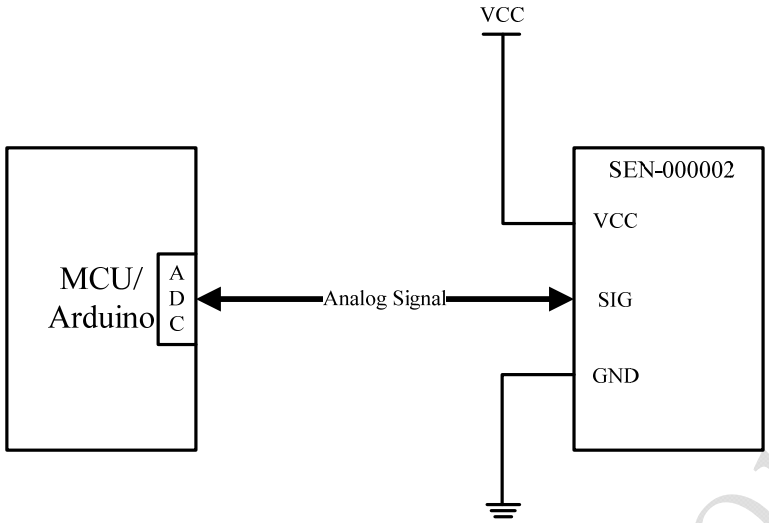

Figure 1-1, Typical Application Schematics

Operation

The protection resistor (4.7Kohms) and the adjustable (0-50Kohms) are in serial which forms a load resistor R_L (4.7-54.7Kohms). The sensor's resistance R_S and R_L forms a voltage divider. The output voltage on the signal pin could be read by Arduino or MCU via ADC. Given a value of R_L , Power Supply Voltage, and output voltage, R_S could be derived. Based on the chart provided in the MQ-2 datasheet, R_S in clean air under given temperature and humidity is a constant，which is the "initial" resistance of the sensor named R_O. R_O of the resistor could be derived from R_S. The main job of the calibration is to calculate the R_O by sampling and averaging the readings when the module is placed in the clean air. Once the R_O is derived, the concentration of target gas could be calculated by using the R_S/R_O ratio as the input. To achieve more accuracy, a segmented look-up table should be used. However, in the demonstration, a linear formula is used as an approximation to the original curve.

Sample Code for Arduino

```
/*******************Demo for MQ-2 Gas Sensor Module V1.0*****************************
Author:  Tiequan Shao: tiequan.shao@sandboxelectronics.com
         Peng Wei:     peng.wei@sandboxelectronics.com

Lisence: Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0)

Note:    This piece of source code is supposed to be used as a demostration ONLY. More
         sophisticated calibration is required for industrial field application.

                                                    Sandbox Electronics    2011-04-25
************************************************************************************/

/************************Hardware Related Macros*********************************/
#define         MQ_PIN                       (0)     //define which analog input channel you are going to use
#define         RL_VALUE                     (5)     //define the load resistance on the board, in kilo ohms
#define         RO_CLEAN_AIR_FACTOR          (9.83)  //RO_CLEAR_AIR_FACTOR=(Sensor resistance in clean air)/RO,
                                                     //which is derived from the chart in datasheet

/***********************Software Related Macros************************************/
#define         CALIBARAION_SAMPLE_TIMES     (50)    //define how many samples you are going to take in the calibration phase
#define         CALIBRATION_SAMPLE_INTERVAL  (500)   //define the time interal(in milisecond) between each samples in the
                                                     //cablibration phase
#define         READ_SAMPLE_INTERVAL         (50)    //define how many samples you are going to take in normal operation
#define         READ_SAMPLE_TIMES            (5)     //define the time interal(in milisecond) between each samples in
                                                     //normal operation

/**********************Application Related Macros**********************************/
#define         GAS_LPG                      (0)
#define         GAS_CO                       (1)
#define         GAS_SMOKE                    (2)


/*****************************Globals***********************************************/
float           LPGCurve[3]  =  {2.3,0.21,-0.47};   //two points are taken from the curve.
                                                    //with these two points, a line is formed which is "approximately equivalent"
                                                    //to the original curve.
                                                    //data format:{ x, y, slope}; point1: (lg200, 0.21), point2: (lg10000, -0.59)
float           COCurve[3]  =  {2.3,0.72,-0.34};    //two points are taken from the curve.
                                                    //with these two points, a line is formed which is "approximately equivalent"
                                                    //to the original curve.
                                                    //data format:{ x, y, slope}; point1: (lg200, 0.72), point2: (lg10000,  0.15)
float           SmokeCurve[3] ={2.3,0.53,-0.44};    //two points are taken from the curve.
                                                    //with these two points, a line is formed which is "approximately equivalent"
                                                    //to the original curve.
                                                    //data format:{ x, y, slope}; point1: (lg200, 0.53), point2: (lg10000,  -0.22)
float           Ro           =  10;                 //Ro is initialized to 10 kilo ohms

void setup()
{
  Serial.begin(9600);                               //UART setup, baudrate = 9600bps
  Serial.print("Calibrating...\n");
  Ro = MQCalibration(MQ_PIN);                       //Calibrating the sensor. Please make sure the sensor is in clean air
                                                    //when you perform the calibration
  Serial.print("Calibration is done...\n");
  Serial.print("Ro=");
  Serial.print(Ro);
  Serial.print("kohm");
  Serial.print("\n");
}

void loop()
{
   Serial.print("LPG:");
   Serial.print(MQGetGasPercentage(MQRead(MQ_PIN)/Ro,GAS_LPG) );
   Serial.print( "ppm" );
   Serial.print("    ");
   Serial.print("CO:");
   Serial.print(MQGetGasPercentage(MQRead(MQ_PIN)/Ro,GAS_CO) );
   Serial.print( "ppm" );
   Serial.print("    ");
   Serial.print("SMOKE:");
   Serial.print(MQGetGasPercentage(MQRead(MQ_PIN)/Ro,GAS_SMOKE) );
   Serial.print( "ppm" );
   Serial.print("\n");
   delay(200);
}

/****************** MQResistanceCalculation *************************************
Input:   raw_adc - raw value read from adc, which represents the voltage
Output:  the calculated sensor resistance
Remarks: The sensor and the load resistor forms a voltage divider. Given the voltage
         across the load resistor and its resistance, the resistance of the sensor
```

```
         could be derived.
************************************************************************************/
float MQResistanceCalculation(int raw_adc)
{
  return ( ((float)RL_VALUE*(1023-raw_adc)/raw_adc));
}


/***************************** MQCalibration ****************************************
Input:   mq_pin - analog channel
Output:  Ro of the sensor
Remarks: This function assumes that the sensor is in clean air. It use
         MQResistanceCalculation to calculates the sensor resistance in clean air
         and then divides it with RO_CLEAN_AIR_FACTOR. RO_CLEAN_AIR_FACTOR is about
         10, which differs slightly between different sensors.
************************************************************************************/
float MQCalibration(int mq_pin)
{
  int i;
  float val=0;

  for (i=0;i<CALIBARAION_SAMPLE_TIMES;i++) {            //take multiple samples
    val += MQResistanceCalculation(analogRead(mq_pin));
    delay(CALIBRATION_SAMPLE_INTERVAL);
  }
  val = val/CALIBARAION_SAMPLE_TIMES;                   //calculate the average value

  val = val/RO_CLEAN_AIR_FACTOR;                        //divided by RO_CLEAN_AIR_FACTOR yields the Ro
                                                        //according to the chart in the datasheet

  return val;
}
/*****************************  MQRead *********************************************
Input:   mq_pin - analog channel
Output:  Rs of the sensor
Remarks: This function use MQResistanceCalculation to caculate the sensor resistenc (Rs).
         The Rs changes as the sensor is in the different consentration of the target
         gas. The sample times and the time interval between samples could be configured
         by changing the definition of the macros.
************************************************************************************/
float MQRead(int mq_pin)
{
  int i;
  float rs=0;

  for (i=0;i<READ_SAMPLE_TIMES;i++) {
    rs += MQResistanceCalculation(analogRead(mq_pin));
    delay(READ_SAMPLE_INTERVAL);
  }

  rs = rs/READ_SAMPLE_TIMES;

  return rs;
}

/*****************************  MQGetGasPercentage **********************************
Input:   rs_ro_ratio - Rs divided by Ro
         gas_id      - target gas type
Output:  ppm of the target gas
Remarks: This function passes different curves to the MQGetPercentage function which
         calculates the ppm (parts per million) of the target gas.
************************************************************************************/
int MQGetGasPercentage(float rs_ro_ratio, int gas_id)
{
  if ( gas_id == GAS_LPG ) {
     return MQGetPercentage(rs_ro_ratio,LPGCurve);
  } else if ( gas_id == GAS_CO ) {
     return MQGetPercentage(rs_ro_ratio,COCurve);
  } else if ( gas_id == GAS_SMOKE ) {
     return MQGetPercentage(rs_ro_ratio,SmokeCurve);
  }

  return 0;
}

/*****************************  MQGetPercentage **********************************
Input:   rs_ro_ratio - Rs divided by Ro
         pcurve      - pointer to the curve of the target gas
Output:  ppm of the target gas
Remarks: By using the slope and a point of the line. The x(logarithmic value of ppm)
         of the line could be derived if y(rs_ro_ratio) is provided. As it is a
         logarithmic coordinate, power of 10 is used to convert the result to non-logarithmic
         value.
************************************************************************************/
int  MQGetPercentage(float rs_ro_ratio, float *pcurve)
{
  return (pow(10,( ((log(rs_ro_ratio)-pcurve[1])/pcurve[2]) + pcurve[0])));
```

}

Demo Output

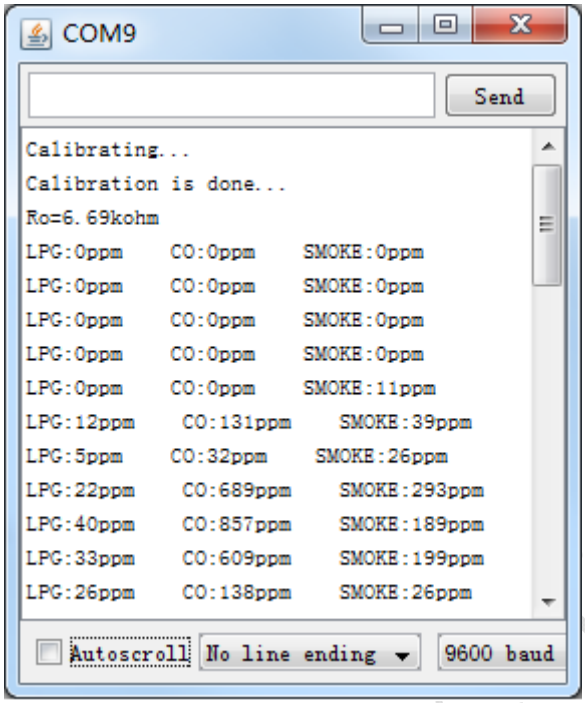

Figure 1-2 Demo output as the sensor approaches a lit cigarette

Dimensions