Gas Alarm
Project number F175

Abstract

The article describes the home combustible gas detector. Main motivation for this construction was my 3 years old son who has very “positive” relationship with the home hardware. Of course, nice buttons on the gas cooker are very interesting for him and he can hear nice hiss of the gas. The first situation like this accelerated my plan to develop some gas detector. The detector can be used in the kitchen, near gas boiler, in the garage, during camping etc. Everywhere where leaking gas could make problems. By changing the sensor type, the detector can be used for other gas types. Used sensor TGS2611 is recommended for methane and natural gas detection.

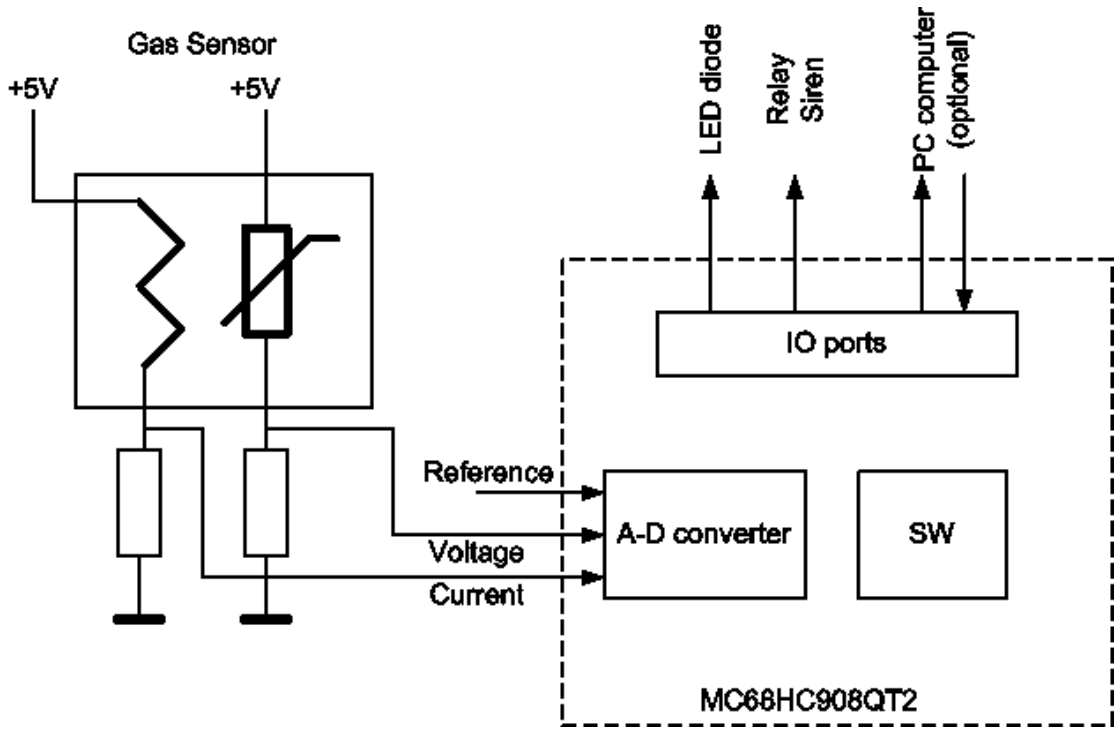
The detector is based on the commercial gas sensor from Figaro company. The electronic circuit needs to evaluate change in the internal sensor resistance. For proper operation you also need to use few comparators what watch the reference voltage, sensor resistance and sensor proper operation. You need to add delay after power on and the delay after first gas detection to avoid false alarms. If you want to detect two different gas concentrations you need to add additional comparator. The result is at least two DIL14 quad comparators, some transistors and many resistors.

I have decided to use MC68HC908QT2 “Nitron” microcontroller because it makes the detector much simpler. It is in small DIP8 (or SO8) package and 4-inputs 8-bit analog to digital converter covers all demands for this application. All delays and comparators are implemented in the software. The result is small, cost effective and flexible gas detector. It also has possibility to upgrade the firmware through serial link from PC using poor man’s interface (zener diode and the resistor). The total cost of the application is lower compared to standard operational amplifier solution. This simple application does not need external oscillator, the internal Nitron’s oscillator is used. It enables to use pins OSC1 and OSC2 as AD converter inputs.

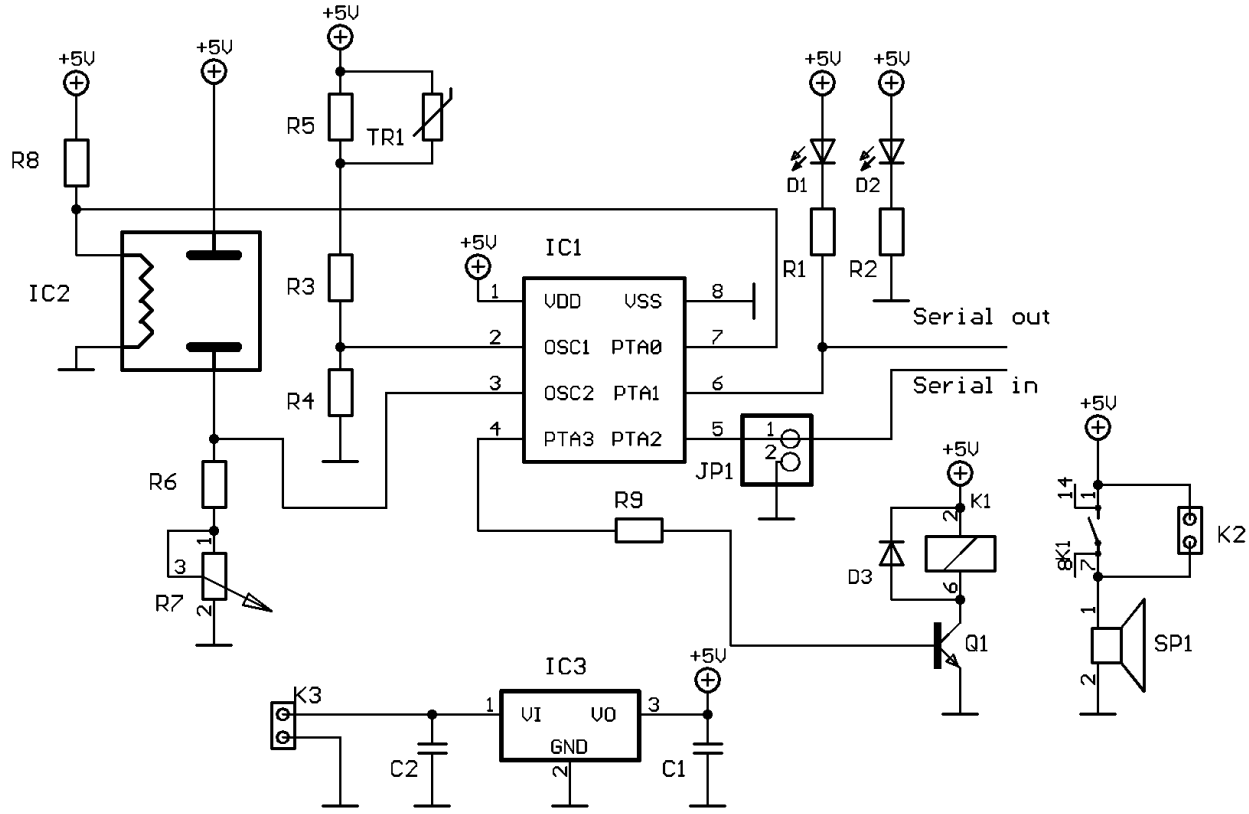
The software is written in the special, free version of the CodeWarrior. The code is uploaded to the chip using bootloader method described in AN2295. This allows really low cost and easy development of any Nitron application. The program measures voltage across the sensor and compares it to the reference. If this value is higher then the threshold value for more then 15 sec, level 1 alarm is initiated. If the value exceeds the second threshold, level 2 alarm starts. Alarm levels are indicated by the LED diode, by the piezzo siren and also relay contacts are closed. The relay can be used for switching another necessary device like home alarm or the fan. If the gas concentration falls down, alarm is stopped. With the memory function the detector will indicate alarm continuously and the user will see potential risk of the gas leakage.

The software also controls current consumption of the sensor heather and the value of the sensor resistance. If any of these values is out of limits, the detector indicates sensor malfunction. The software can be simply modified for other functions based on user wish.

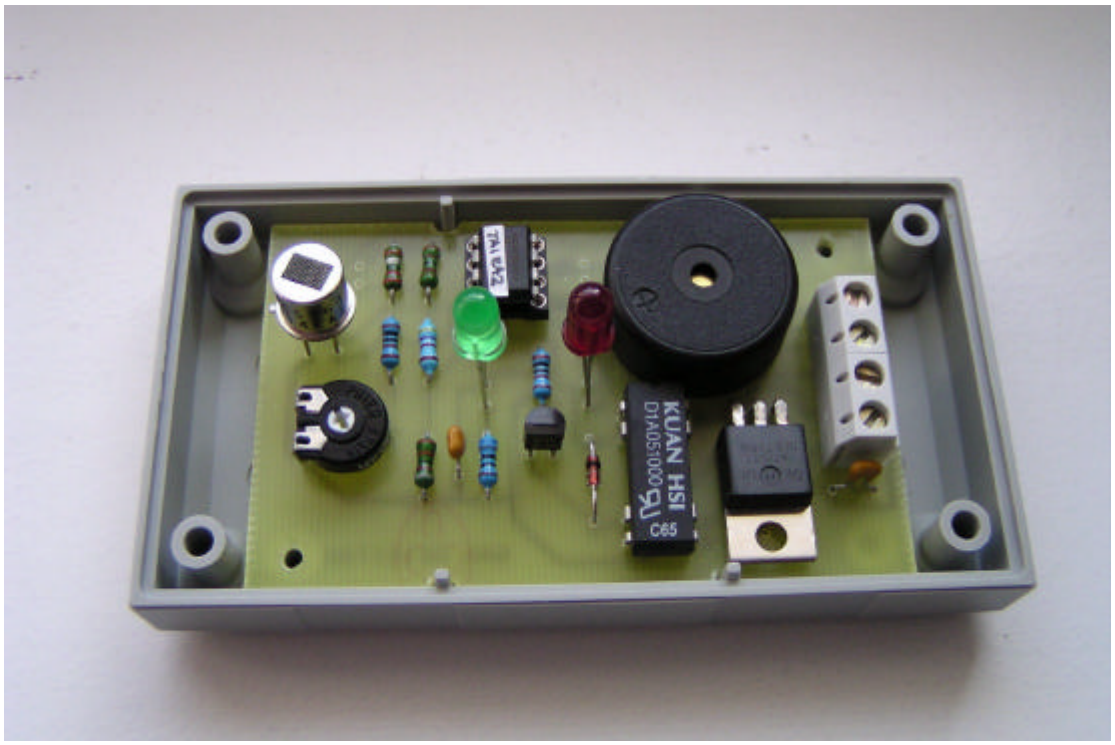
The detector is built in the Bopla EG 1030 L box and is powered from external power supply. This enables to use certified power supplies for given AC voltage (110V or 230V).



Gas Detector block diagram.



Detector schematic.



Detector photo, inside.

Detector photo, overall view.

Main source code

```
/*
The programm for gas detector

Project number F175

*/


#include "map.h"            // processor register defines
#include "sci.h"            // Bootloader's SCI included


#define BYTE unsigned char
#define WORD unsigned int

#define EnableInterrupts()  asm CLI
#define DisableInterrupts() asm SEI


#define Gas 2
#define Reference 3
#define Consumption 0               // number of ADC input for given value

#define Alarm2Level 20                      // delta between alarm1 and alarm2 levels
#define Hysteresis 5                        // hysteresis on measurement
#define CurrentLimit 252                            // voltage level for sensor
current problem
#define WaitTime 2                                  // how long time I need signal
before alarm
#define SensKOLow 10                        // below this value the sensor is bad


////////////////////////////////////////////////
// I/O mappings

#define LED    PTA1
#define OUTPUT         PTA3
#define MEMORY_FUN     PTA2

////////////////////////////////////////////////

#pragma DATA_SEG SHORT MY_ZEROPAGE

WORD Tick1;
BYTE Tick2,Led_on;
unsigned int x;
unsigned int i;

#pragma DATA_SEG DEFAULT

BYTE ConsumptionValue, GasValue;
BYTE Alarm1, Alarm2, SensorAlarm;
BYTE ReferenceValue;
BYTE Al_enable;

void Init(void)
{
#ifdef COP_ENABLE
    CONFIG1 = 0x00;
#else
    CONFIG1 = CONFIG1_COPD;
#endif

    DDRA = 0xFF;                    // all outputs
    DDRA_BIT0 = 0;                  // PTA0 is input
    DDRA_BIT4 = 0;                  // PTA4 is input
    DDRA_BIT5 = 0;                  // PTA5 is input
    PTAPUE = 0x00;                  // no pull ups

    TSC_PS2 = 1;
    TSC_PS1 = 1;
    TSC_PS0 = 0;                    // prescaller /64
```

```
        TSC_TOIE = 1;                // enables TIM interrupt
        TSC_TSTOP = 0;               // runs timer

    TMODH = 0xC3;
    TMODL = 0x50;                    // 1sec preset

    EnableInterrupts();


}

interrupt IV_IRQ1 void Int_IRQ(void)
{
//    ISCR_ACK1 = 1;              /* IRQ ack */
}

interrupt IV_TCHO void Int_TimerCH0(void)
{
//    TSC0;
//    TSC0_CH0F = 0;           /* clearing CH0 flag */

}

interrupt IV_TCH1 void Int_TimerCH1(void)
{
//    TSC1;
//    TSC1_CH1F = 0;           /* clearing CH1 flag */
}

interrupt IV_TOVF void Int_TimerTOV(void)
{
    TSC;
    TSC_TOF = 0;              /* clearing TOV flag */
//    TMODH = 0xC3;
  //  TMODL = 0x50;                  //1sec preset
        TMOD = 500;
        Tick1++;
        Tick2++;

        if (Al_enable == 1)
        {
                if (Tick2 < 2)
                {
                        LED = 0;              //LED is on
                        OUTPUT = 1;
                }
                else
                {
                        LED = 1;
                        OUTPUT = 0;
                }
                if (Tick2 > Led_on) Tick2 = 0;
        }
}

interrupt IV_KBRD void Int_Keyboard(void)
{
//    KBSCR_ACKK = 1;         /* keyb int. acknowledge */

}

interrupt IV_ADC void Int_ADConv(void)
{
BYTE temp;

//    temp = ADR;             /* just read ADR or write to ADSCR to clear int. */

}


/*-------------------------------------------------------*/

void Waiting(unsigned int Koliko)
{

        Tick1 = 0;
        while(Tick1 != (Koliko*10));
```

```
}


/*-----------------------------------------------------*/

BYTE Measure (BYTE InputNumber)
{

BYTE temp;

       ADSCR = 0x80 | InputNumber;                                  // sets one
conversion, no interrupt
       while(ADSCR_COCO == 0);                                                  //
waits for conversion complete
       temp = ADR;                                                              //
reads value
       return(temp);

}

/*-----------------------------------------------------*/
void main(void)
{
    Init();
    SCIAPIInit();


#ifdef COP_ENABLE
        COPCTL = 0;             /* bump watchdog */
#endif

       LED = 0;                                                 // switch led on
       Waiting (150);                                       // waits 150sec after
reset - sensor needs it
       SensorAlarm = 0;

       for (;;)                                                         // stadard
operation, wait for Ticks and then measure&write
       {
              GasValue = Measure(Gas);
              ReferenceValue = Measure(Reference);
              ConsumptionValue = Measure(Consumption);

              if (GasValue < (ReferenceValue - Hysteresis))
              {
                     Alarm1 = 0;
                     Alarm2 = 0;                            // delay alarms, level is
below treshold
              }
              if (GasValue > ReferenceValue)
              {
                     Al_enable = 1;
                     if (GasValue > (ReferenceValue + Hysteresis))
                     {
                            Alarm2 = 1;                         // 2nd alarm level,
big concentration
                            Alarm1 = 0;
                     }
                     else
                     {
                            Alarm2 = 0;
                            Alarm1 = 1;                                 // alarm 1
level
                     }
              }
              if (ConsumptionValue > CurrentLimit)
              {
                     SensorAlarm    = 1;                         // sensor alarm!
              }
              else if (GasValue < SensKOLow)
              {
                     SensorAlarm    = 1;                         // sensor alarm!
              }
              else SensorAlarm = 0;
```

```
                if ((SensorAlarm == 0) & ((Alarm1 == 0) & (Alarm2 == 0)))
                {
                        if (MEMORY_FUN)                                 // if mem_fun pin
is open, then delete alarms
                        {
                                Tick1 = 0;                              // no alarm, clear
flags
                                LED = 1;
                                OUTPUT = 0;
                                Al_enable = 0;
                        }
                }

                else if (Tick1 >= WaitTime)
                {
                        if (Alarm1)    Led_on = 25;
                        if (Alarm2)    Led_on = 10;
                        if (SensorAlarm) Led_on = 1;
                }

                Waiting (1);

        }

}
```