

smartMODUL™

Manual



Contents

1. General	Page 3
2. Installation of the smartMODUL™	Page 3
3. Assignment and properties on the connecting pins on smartMODUL™	Page 4
4. Reading the smartMODUL™ via Pin OUT 1 (TTL Digital Out)	Page 6
5. smartMODUL™ hardware implementation	Page 8
a. Connecting to the RS232-interface of apc	Page 8
b. Connecting to a μ -Controller	Page 10
6. Register connections of the smartMODUL™	Page 11
7. Communication with smartMODUL™ via Modbus-Open-Protocol	Page 13
8. Examples for the Communication with smartMODUL™	Page 16
a. Example of communication via the hardware UART of a μ -Controllers	Page 16
b. Example of communication via the software UART of a μ -Controllers	Page 18
c. Example for the check sum calculation in C	Page 19
d. Example for data sending in DELPHI	Page 21
Technical Data	Page 22

1. General

The smartMODUL™ is a self-sufficient gas measuring sensor that due to its convenient interfaces can be easily and quickly integrated into existing measurement and control systems for gases.

A measurement technique based on the infrared absorption offers both selectivity and reliable precise measurement.

The device's compact size and minimal maintenance requirements mean its ideal for use in difficult situations.

2. Installation of the smartMODUL™

Caution: Infrared-gas sensors are optical measurement systems. Because of this, heavy crushes and abrasion should be avoided!

The smartMODUL™ commands two mounting straps on the sides. With these mounting straps, the smartMODUL™ can be screwed. Please use only cylinder head screws and in no case use counter-sunk-screws. The maximum fastening torque for the screws on the sensor is 0,3Nm. Because of the humble fastening torque it is advisable, to use a bolt lock like Loctite (middle hard) or an annular gear in each case.



Figure 1: Mounting straps

At the installation, please consider that no mechanical pressure is exerted on the housing in the location. Besides, the smartMODUL™ has to be fixed on a flat area, so that the housing is not wedged.

The use of aggressive cleansers and adhesives has to be avoided.

3. Assignment and properties of the connecting pins on smartMODUL™:

There are two possible pin combinations for smartMODUL™ connector plug: a 4-pin and 7-pin version.

The following section describes the 7-pin version as the first 4 pins are connected in exactly the same manner as in the 4-pin version.

-
- OUT 2** → Device fault (active o). Open collector output for the display of device faults, e.g. radiation source failure.
- OUT 3** → Main Gas Alarm (active o). Open collector output for gas main alarm display. The trigger threshold is set using the software.
- OUT 4** → Gas Pre-Alarm (active o). Open collector output for gas pre-alarm display. The trigger threshold is set using the software.

Note: *The pins OUT1-4 are open collector outputs. They switch again GND and may be powered with max. 100mA! The voltage may not be more than 30V. When driven an inductor a fly back diode is necessary!*

4. Reading the smartMODUL™ via Pin OUT 1 (TTL Digital Out):

Displaying the smartMODUL™ TTL-signal:

The smartMODUL™ can display up to 7 different statuses via pin OUT 1, described in the following section.(active = 0, inactive = 1)

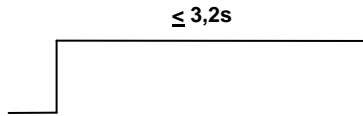
- All OK 3.2s inactive
- Gas Pre-Alarm 0.8s active / 2.4s inactive
- Main Gas Alarm 1.2s active / 2.0s inactive
- Inc – Alarm 1.6s active / 1.6s inactive (optionally set)
- Ex – Alarm 2.0s active / 1.2s inactive (optionally set)
- Tox – Alarm 2.4s active / 0.8s inactive (optionally set)
- Device Fault 3.2s active

As you can see the different alarm statuses are displayed. Switch status is active 0.
Alarm Inc., Ex. and Tox. can be optionally set alarm levels and are not included in the standard smartMODUL™ set up.

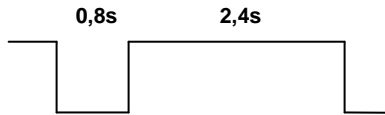
To visualize the TTL signal on an oscilloscope, a pull-down resistance of 2kOhm needs to be patched to V+.

The signal are then displayed as a voltage trace.

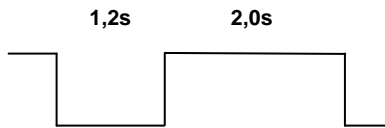
All ok



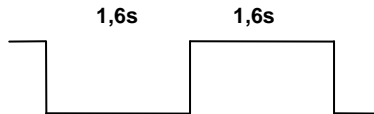
Gas Pre-Alarm



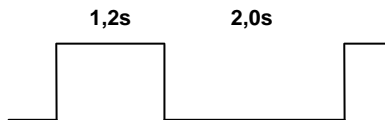
Main Gas Alarm



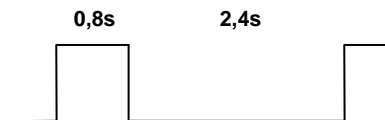
Inc. Alarm
(Optional)



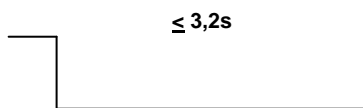
Ex. Alarm
(Optional)



Tox. Alarm
(Optional)



Device Fault



5. smartMODULS™ hardware implementation

Before describing the Modbus protocol and the individual smartMODULS™ registries in greater detail, the following section describes how to connect a module to a PC RS232 interface or μ -Controller.

5.a Connecting to the RS232-interface of a PC

There are two different options for connecting a smartMODULS™ to a PC via a RS232 interface: either using a small additional switch enables bi-directional data transfer, or a level controller.

Level controller

With a level controller, the interface uses TTL level for communication. As the send and receive paths in the smartMODULS™ are combined, so must those in the level controller to which it is connected also be combined. This is most easily done by inserting a 680 Ω resistance into the send path of the level controller. The RX – path should then be used as a common path to the module.

Connection using an additional switch

The circuit shown in Figure 3 is a RS232 interface which does not work with level signal translation, but directly with the TTL signal. Nevertheless, it is possible to connect the switch outputs directly to the PC.

The send and receive paths are combined as a single path. This is possible since communication is exclusively initialized by the master.

If smartMODUL™ is not directly addressed, it remains on receive.

Warning: signals sent from the PC are also always received by them. This should be taken into account when setting the system up.

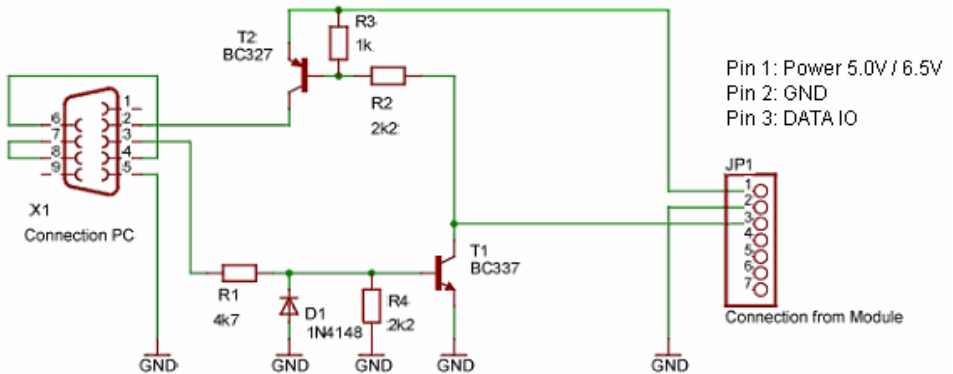


Figure 4. Hardware for communication via a RS232-interface. **WARNING: Pin 1 is connected with supply voltage!!! (see type plate)**

RS232-settings at the PC

Baud rate: 2400
Data bits: 7
Stop bits: 1
Parity: Even
Timeout: 1000ms
Retries: 3

5.b Connection to a μ -Controller

As an alternative to communication via the PC's RS232 interface, the smart*MODUL™* can be connected directly to a μ -controller.

Hardware UART (Universal Asynchronous Receiver Transmitter)

It is probably easiest to use existing UART hardware and the send and receive paths. As the send path is generally high impedance this should pose no problem. However, the latter must be switchable.

If the controller is programmed in-circuit and the port pins can functionally adopt a non-defined state, then a protective resistance should be introduced in the receive path to limit current.

The RS232 interface hardware includes a buffer in which data for dispatch is written. When the required data is written a flag is set in the μ -controller's UART status register. In the example described in Chapter 8, Section a, a request is addressed to this flag so that the data is read at the correct time.

UART software

The RS 232 interface software has no buffer and must be directly read. An additional problem is the kbhit() command that looks to see if there is a character on the receive path. Implementation of the software interface version requires multiple keyboard strokes increasing the chance of error input.

In addition, to set up a software UART, it is only possible to connect the data output of the smart*MODUL™* to a single pin of the μ -controller. This is possible via a serial half-duplex connection, described in greater detail in Chapter 7.

The example in Chapter 8, Section b, describes how these errors can be detected.

RS232 settings for the μ -controller

Baud rate:	2400
Data bits:	7
Stop bits:	1
Parity:	Even
Timeout:	1000ms
Retries:	3

6. Register connections for the smartMODUL™

In the version described here the register is (in the same sequence as in the Host SW)

0xC0	Modbus_address	Current Modbus address of smartMODUL™ The address can be read and written
0x80 – 0x83	DeviceType	The type of device connected. Read only.
0x86 – 0x89	SerialNr	The device serial number. Read only.
0x84 – 0x85	SoftwareVersion	Software version of device connected. Read only.
0x05	MOD	Assumed value for the internal calculation of the concentration. Read only
0x0A	Concentration	This register shows the gas concentration as a value.

The individual flags from right to left mean:

Flag 0:	Testflag,	Value 1 with a device test
Flag 1:	Warmup,	Value 1 after start, approx 10s.
Flag 2:	Syserr,	Value 1 with device fault.
Flag 3:	Alarm,	Value 1 with main gas alarm.
Flag 4:	Warn,	Value 1 with gas pre-alarm.
Flag 5:	Startup,	Value 1 in run-up phase, approx. 90s.
Flag 6:	Korr,	Value 1 when smart <i>MODUL™</i> is temperature compensated.
Flag 7:	mw_ok,	Value 1 when smart <i>MODUL™</i> has been calibrated.

Flags 6 (Korr) and 7 (mw_ok) are internal flags, set during manufacture of each smart*MODUL™*. They are designed for quality control and are set to Value 1 when smart*MODUL™* is temperature compensated and has been calibrated.

Example:

You send to the smartMODUL the string:

:A00300090001A2 (Check Sum A2)

And you get back the answer:

:A0030200CoF7 (Check Sum F7)

: → Start
A0 → Address 160
03 → Read out
02 → No. of bytes
00Co → Register Data "Co" hex. = 192 dec. = 11000000 bin.
Only flag 6 and flag 7 are set to "1". The module works normal.

7. Communication with the smartMODUL™ via the Modbus-Open-Protocol:

Reading the TTL signal provides the user with only a fraction of the information available from the smartMODUL™.

As the smartMODUL™ has large amount of further information available, use of a BUS protocol makes good sense.

In general, the Modbus-Protocol operates after the Master/Slave principle. The Master (PC or μ -controller) sends a request to Slave (smartMODUL™), which answers. The constancy of the phase, until all data are received, depends on how many registers are elected. Basically, it can be said, that the smartMODUL™ reacts within 100 ms on the request. Afterwards, the characterband is directly sent without reply break. The Slave does not send data without request. The request is always interpreted after the sending of CRLF.

When an incomplete request is sent, the smartMODUL™ does not reply.

smartMODUL™ uses a modified form of Modbus Open Protocol. This is different from the standard version in that only one path is used for send and receive.

This ASCII protocol uses a serial half-duplex connection.

Data telegram structure

The following section describes how a data string requests to the smartMODUL™ are structured. The example shows the current modulation of a smartMODUL™ with an address of 160.

The string looks like the following:

:A00300050001A60DoA

Start	Adress	Control command	Data	Check sum LRC	End
1 character	2 characters	2 characters	0-100 * 2 characters	2 characters	CR LF
:	A0	03	00 05 00 01	A6	0D 0A

Comment: Addresses, control commands and data are prefixed with "0x" and the actual address / command "nn". The "0x" indicates that data transmission is hexadecimal. As the Modbus Open Protocol is hexadecimally defined ASCII this information is superfluous and only the address or command is transmitted. The string contains no "0x": from "0x05" results in just "05".

Start

Basically the data telegram starts with a colon ":".
It is irrelevant whether it is request or answer.

Address

This defines to which device the string is directed.

The example shows a request to a CO₂ - smart*MODUL*TM with a standard address of 160, (0xA0→A0).

To search for the addresses of the modules, these are connected. Now, any register (e.g. concentration) of all module addresses (1-255) can be requested with a timeout within 1 second. When a module is activated with the correct address, it reacts by sending an answer. In this answer, the module address is included. Therefore, at the end of the search cycle, it can be analyzed which module addresses are currently connected to the Bus system.

Control command

The control command indicates what exactly should occur at the given address. Essentially the smart*MODUL*TM distinguishes between "Read→0x03" and "Write→0x06".

The command in the example shown is "Read register" (0x03→03)

Data

The number of registers is also sent as a parameter in the data. The example this is "Start Address High (0x00→00) / Low (0x05→05)" and "Number Register High (0x00→00) / Low (0x01→01)".

The "Start Address High" and "Start Address Low" together indicate the register address to which the control command is directed; in this case address 0005→0x05 "MOD".

The "Number Register High" and "Number Register Low" state how many registers should be read along with the start address. Should 10 registers be read, then this number should be entered as 0010.

Registers 05 to 14 are then read and transmitted in sequence.

Transmission of individual data is in hexadecimal format!

This in turn results in a doubling of bytes transmitted when converting from ASCII to Hex.

Check sum

The check sum transmitted is calculated according to the LRC method, from all bytes sent without CR and LF characters.

Bytes are summed and this total subtracted from 0xFF.

A "1" is added to this result, making LRC complete.

The value shown in the example is "A6".

The check sum is always transmitted with the data and recalculated by the recipient. Should a value in the data set become corrupt, then the check sum calculated by the recipient would be different from that sent. The data set in this case would be unusable.

An answer to the string shown above would like look:

:A003020001090DoA

- : → Start
- A0: → Device address
- 03: → Control command (read)
- 02: → Number of the register to which the answer is transferred
- 00: → 1. Part of the answer, here value 0
- 01: → 2.Part of the answer, here value 1
- 09: → Check sum of transferred packets
- 0D: → CR
- 0A: → LF

A modulation with the value of 1 has been read off from register 05 and transmitted as answer. In Chapter 8, Section c a programmed example of the calculation of the check sum is shown.

8. Examples for the communication with the smartMODUL™

8.a Example of communication via the hardware UART of a μ -controller

In the example, a PIC 16F874 with 20 MHz is used and the check sum not computed. In addition, a variant without interrupt is described.

The programme waits for a status flag to be set in the μ -controller that indicates all the data has been sent via the send path.

```
/** definition of the interface */
#include <rs232.h>
/** prepare request to the status flags */
#define TXSTA 0x98 //from Datenblatt
#define TRMT  TXSTA.1
#define wait_for_RS232() while(TRMT == 0)
void main(void) { // main function
  boolean d = 1;
  boolean e = 1;
  char c1;
  int i;

  while (TRUE) {
    i = 0;
    e = 1;
    d = 1;
    delay_ms(100);
    fprintf(CO, "A00300050001A6\r\n"); //send request
    wait_for_RS232(); //wait for all data to be sent
    if (kbhit(CO))
    {
      /** wait for the start of the strings to be received */
      while (e)
      {
        if(fgetc(CO) == ':') e=0;
        delay_us(2);
      }
      while (d) //running loop until LF
      {
        c1=fgetc(CO); //get characters from the interface
        str[i++]=c1; //store characters in the string
      }
    }
  }
}
```

```
        if(c1 == 10) d = 0;           // 10 is the ASCII character for LF
        }
    auswerten();                     // modulation
    }
}

void auswerten()
{
    long co2long, co2r;
    int hilf;

    costring[0]='0';                 // simulates hexadecimal number
    costring[1]='x';

    for(hilf=7;hilf<11;hilf++)       //saves relevant values for the CO2 value
    {
        costring[hilf-5]=str[hilf];
    }
    costring[6]=0;                   //end sign in the string gets set
    co2long = atol(costring);        //changes string to long

    co2r=0.0000024169*(co2long*co2long*co2long); // calibration
    co2r=co2r+(0.001064478*(co2long*co2long));
    co2r=co2r+(1.5139504215*(co2long));

    i=0;                             //reset of the variable
    for(hilf=0;hilf<20;hilf++) str[hilf]=0;
}
```

8.b Example of communication via software UART of a μ -Controllers

In the example PIC 16F874 with 20 MHz is used and the check sum not computed. The programme waits until the start character of the received string has been read at the interface. During this period the controller can undertake no other processes, a situation impossible for time-critical applications (USB connections).

```
/** definition of the interface */
#include <rs232(baud=2400,XMIT=PIN_Do,rcv=PIN_D1,STREAM=CO,bits=7,parity=E,errors)

void main(void) { // main function
  boolean d = 1;
  boolean e = 1;
  char c1;
  int i;

  while (TRUE) {
    i = 0;
    e = 1;
    d = 1;
    delay_ms(100);
    fprintf(CO,":A003000500001A6\r\n"); //send request

    /** wait for the start of the string to be received */
    while (e)
    {
      if(fgetc(CO) == ':') e=0;
      delay_us(2);
    }

    while (d // running loop until LF
    {
      c1=fgetc(CO); // get characters from the interface
      str[i++]=c1; // store characters in the string
      if(c1 == 10) d=0;
    }

    auswerten(); //see cahpter 8.a
  }
}
```

8.c Example for the check sum calculation in C

(Read out only register 05 → Modulation MOD)

```
Daten[0]=':'; Daten[1]='A'; Daten[2]='0'; Daten[3]='0'; Daten[4]='3';  
Daten[5]='0'; Daten[6]='0'; Daten[7]='0'; Daten[8]='5'; Daten[9]='0';  
Daten[10]='0'; Daten[11]='0'; Daten[12]='1'  
Laenge=12;
```

Regard: *CRLF and LRC are not part of the datas. So they will not be included in the calculation!*

1. Lrc=0; // (Check sum reset)
For(i=1;i<Laenge;i++)
2. Lrc=Lrc+daten[i]; // (All transferred bytes will be added binary (8 bit).
Example: 200 + 200 = 400. Possible with 8 Bit max. 256
→ 144 gets written = LRC.
(In the example above (read MOD) the LRC rest sum is 90!)
3. Lrc=0xFF-Lrc; // (255 - 90 = 165)
4. Lrc=Lrc+1; // (165 + 1 = 166 = A6 in hex. (Check sum))

Example:

String : **A00300050001**

All transferred to ASCII values:

A = 65
0 = 48
1 = 49
3 = 51
5 = 53

Calculation: A+0+0+3+0+0+0+5+0+0+0+1
65+48+48+51+48+48+48+53+48+48+48+49=602
602-256=346
346-256=90 (Rest!)

Check Sum: 255-90+1=166 = A6 in hex.

After the calculation of the check sum the following would be sent:

:A00300050001A60DoA

The following answer could result for example:

:A003020020080DoA (check sum 08 of the data package from the answer)

The structure is thus already written:

: Start of the Frames
A0 -> Adress sender
03 -> Register Data
02 -> No. bytes (HEX!)
0020 -> RegisterData
08 -> Check Sum
oDoA -> CR LF

8.d Example for data sending in DELPHI:

```
procedure modbus_senden(str: string);
var
  lrci: byte;
  count: Integer;
  I, J, K: Integer;
begin
  if comactive=false then begin
    lrci:=0;
    count:=length(str);
    for i:=1 to count do begin
      lrci:=lrci + ord(str[i]);
    end;
    lrci:=255-lrci;
    lrci:=lrci+1;
    logit('MOD-TX: '+str+IntToHex(lrci,2),1);
    str:=''+str+IntToHex(lrci,2);
    sent:=str;
    form1.comport1.WriteStr(str+chr(13)+chr(10));
    form1.comtimer.enabled:=true;
    comactive:=true;
  end
  else logit('MODBUS Belegt',1);
  end;
  modbus_senden('A003000000A');
  /// recieve part
  logit('RX:'+DateTimeToStr(Now)+'-->'+str,1);
  if (str[1]<>' ') then logit('Kein : am Anfang',1);           // look for : as start
  // LRC check
  lrci:=0;
  count:=length(str);
  for i:=2 to count-2 do begin;
    lrci:=lrci + ord(str[i]);
  end;
  lrci:=255-lrci;
  lrci:=lrci+1;
  if (copy(str,count-1,2)<>IntToHex(lrci,2)) then logit('LRC FEHLER '+copy(str,count-1,2)+' '<>
  '+IntToHex(lrci,2),1);
```

Technical data

Product specifications:

Detection principle:	NDIR (Dual Beam)
Detection ranges:	depending on the version (see list of detectable gases)
Mechanical dimensions:	62 x 37 x 30 mm (L x W x H)

General features:

Response time:	< 25 s diffusion
Accuracy:	± 2 % FS*
Stability:	± 2 % FS* over 12 months
Repeatability:	< 2 % FS*
Linearity error:	< 1 % FS*
Failure control:	via software
Calibration:	Zero and Span via software
Operating temperature:	-10°C to 40°C
Air pressure:	950 to 1050hPa
Humidity:	0% to 95 rel. humidity
Warm up time:	< 2 minutes (start-up) 15 minutes (full specifications)

Communication:

Digital output signal:	- Modbus ASCII via UART - Open Collector (Pre-Alarm, Main-Alarm, SysError) - TTL (Ok, Pre, Main, Inc, Ex, Tox, SysError)
------------------------	--

Electrical data:

Operating voltage: *	5,0V or 6,0V DC* (see type plate)
Input Current:	average value 70mA, max. 130mA

*depending on the version