Modbus solutions from ME Advise

Modbus generals

Modbus is probably the most spread serial protocol in industry. Most devices, being able to communicate serial, talk Modbus.

The communication is master / slave driven. Modbus is capable to run full-duplex RS232 lines and half-duplex RS485 solutions. Also newer implementation variants run via Ethernet & TCP/IP.

One of the best sources for more detailed information's is http://www.Modbus.org/

A typical Modbus telegram shown in hex characters is:

Request from master	**02 03 00 08 00 01 05 FB**
with the answer from slave	**02 03 02 02 01 3C E4**

It's to slave **02** request **03**, to send back from his address **0008,** one word (**00 01**), with checksum **05 FB**, and
from slave **02** answer **03**, with **02** bytes, the values **02 01**, and checksum **3C E4.**

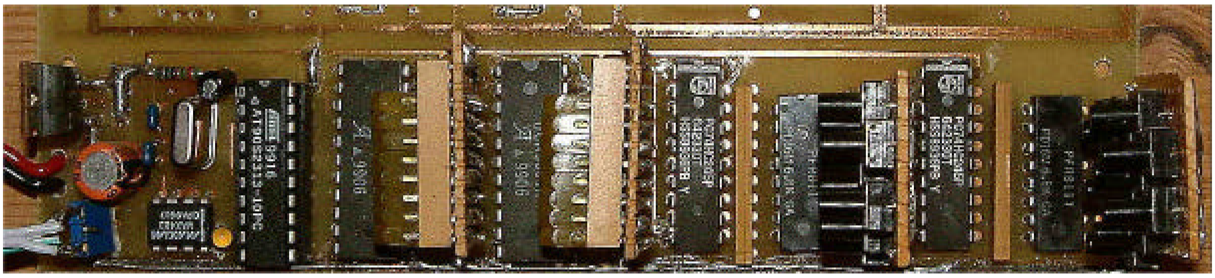
If you want to give your application an easy connectivity to most of the worlds devices, so you should choose Modbus at your first thought.

And if you are a friend of Mark Albert's fine Basic implementation, choose this code to **help yourself saving money**.

The application is purely BASCOM Basic. It's a software implementation for Atmel AVR controllers. The smallest, fully tested version is for AVR 2313 with RS485, running the board you see below.

All here presented modules are a slave implementation, but you can easily change to your specific master implementation.

Modbus slave for AVR 2312



All written in pure **BASCOM Basic**.
Well / fully documented code.

```
' -----[ Constants ]-------------------------------------------------------------

'                            Telegramms solved
'         This device is capable of max 4 bytes data transfer mode 03 & 16 !!!
'
' Request Read    Holding Register(03)  SS 03 SH SL PH PL CL CH
' Answer  Read 16 Holding Register(03)  SS 03 BC 1H 1L CL CH
' Answer  Read 32 Holding Register(03)  SS 03 BC 1H 1L 2H 2L CL CH
'                                       ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦
'                                    DB( 1  2  3  4  5  6  7  8  9 )
'
' Request Preset Single Register(06)    SS 06 RH RL 1H 1L CL CH
' Answer  Preset Single Register(06)    SS 06 RH RL 1H 1L CL CH
'                                       ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦
'                                    DB( 1  2  3  4  5  6  7  8 )
'
' Request Preset Multiple Register(16)  SS 10 SH SL PH PL BC 1H 1L 2H 2L CL CH
' Answer  Preset Multiple Register(16)  SS 10 SH SL PH PL CL CH
'                                       ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦
'                                    DB( 1  2  3  4  5  6  7  8  9  10 11 12 13 )
'
'                                            ' SS = Slave Address
'                                            ' 03 = Command 03
'                                            ' SH = Starting Address High
'                                            ' SL= Starting Address Low
'                                            ' PH = Nr. of Points High
'                                            ' PL = Points Nr.(16 bits) Low
'                                            ' CL = CRC Low
'                                            ' CH = CRC High
'                                            ' BC = Byte Count
'                                            ' 1H = Data 1 High
'                                            ' 1L = Data 1 Low
'                                            ' 2H = Data 2 High
'                                            ' 2L = Data 2 Low

Const Initializing = &HFE                    ' Address to change Myaddress
Const Adressparamter = 10                    ' Slaveparameters position
```

Baudrate 9600 8,N,1. Change baudrate in code prepared.
RS485 control included, RS232 also working.

Telegrams solved: 3, 6 16 = **16 bit** word and **32 bit** long register access possible.

Slave address remote programmable, stored in EERAM.
Always accepted default slave address: **FE** to set address and parameters.

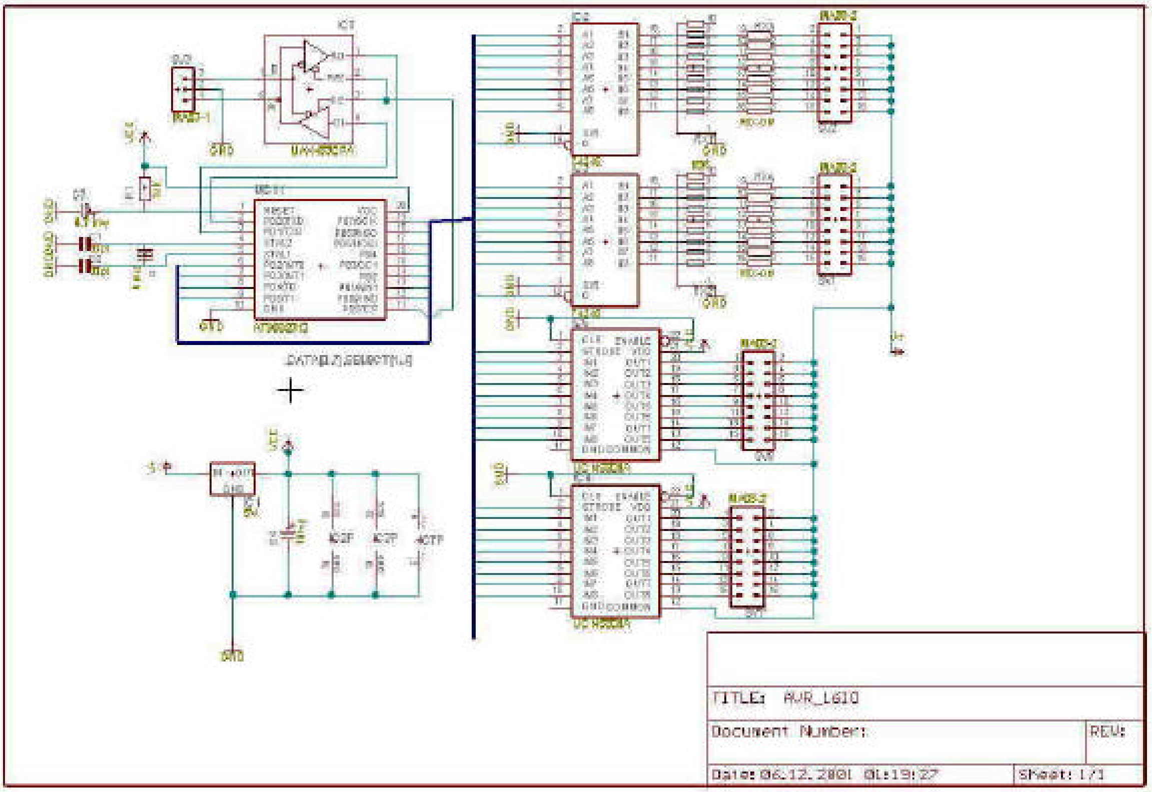
Fully independent **interrupt driven**, send and receive **communication.**

Receive ring buffer.
Independent command detection buffer.
Independent send buffer.

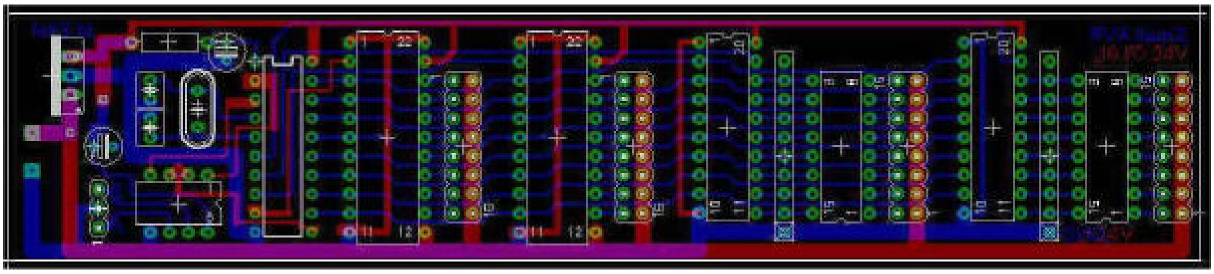
16 Byte Data Array from starting from R0001 implemented.

Time based I/O multiplexing included for 16 in and 16 outputs.

Programmable signal counter for first 8 input pins.



Running layout as base of your development available.



ME Advise

Me, Mike Eitel, I'm in industrial business since 1981 and have done a lot of jobs. When I came in contact with Atmel's AVR, I landed soon at BASCOM. And I'm using that compiler since August 1999 and [as I'm too old to be a genius ;-)] I'm used to write a lot of comments.

Beside of the AVR, I'm a specialist for industrial solutions, especially when they need a control system with high level HMI, or most commonly called SCADA system. Made some nice implementations. For some years I sold a SW called Wizcon, distributed by Axeda www.axeda.com. Still using Wizcon, nothing was more logical, than making a connection between my AVR projects and that PC based visualization. For reasons I explained before, I choose Modbus as best protocol, realize and test my applications that way. I meantime I have a AVR family solution with 8515 with LCD and 8535 with PID regulators. The bigger processors run additionally multitasking. Works fine!

Support email: michael.eitel@schweiz.ch

Address map

Low byte	High byte	Function
00	**01**	16 multiplexed input 1..16 signal status
02	**03**	16 multiplexed outputs, normal function, 1 => output 1..16 goes high
04	**05**	16 same outputs, 1 means out1..16 goes high until input1..16 is high
06	**07**	16 bit counter parametered counting of input 1..8
08	**09**	Counter parameter Slave address
0A	**0B**	Spare
0C	**0D**	Spare
0E	**0F**	Spare

Content

Module / label explanation:

Conditional compiler flags

```
$regfile = "2313def.dat"
```

This is compiled for AVR 2313

```
$lib "Modbus.lbx"
```

The checksum is calculated via marks routine

```
Const Test = 1                  ' When 1 then Test modus
```

Allows working without existing I/O chips

```
Const Stk200 = 1                ' When 1 then STK200 else DTR103
```

This allows working with STK200 and RS232 without having the board.

Define serial communication parameters

```
$baud = 9600                    ' Modbus speed
#if Stk200
   $crystal = 4000000
   Ubrr = 25                    ' For 9600 baud
#else
   $crystal = 8000000
   Ubrr = 51                    ' For 9600 baud
#endif
```

Telegramms solved

This device is capable of max 4 bytes data transfer in mode 03 or mode 16. Otherwise the receive and send buffers are to small

Two sorts of communication are solved:

Reading data in 16 bit and 32 bit request via the telegram **03** and
Sending 16 bit data via telegram **06** and 32 bit via telegram **16**.

```
Request Read      Holding Register(03)  SS 03 SH SL PH PL CL CH
Answer  Read 16   Holding Register(03)  SS 03 BC 1H 1L CL CH
Answer  Read 32   Holding Register(03)  SS 03 BC 1H 1L 2H 2L CL CH
                                        ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦
                                    DB( 1  2  3  4  5  6  7  8  9 )
```

```
Request     Preset Single Register(06)  SS 06 RH RL 1H 1L CL CH
Answer      Preset Single Register(06)  SS 06 RH RL 1H 1L CL CH
                                        ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦
                                    DB( 1  2  3  4  5  6  7  8 )
```

```
Request    Preset Multiple Register(16) SS 10 SH SL PH PL BC 1H 1L 2H 2L CL CH
Answer     Preset Multiple Register(16) SS 10 SH SL PH PL CL CH
                                        ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦  ¦
                                    DB( 1  2  3  4  5  6  7  8  9  10 11 12 13
```

SS = Slave Address
03 = Command 03 as example for a telegram type 03
BC = Byte Count. In some telegrams existing byte counter
SH = Starting Address High of the AVR data area
SL= Starting Address Low
PH = Number of Points High or general spoken number of requested word data’s
PL = Points Nr.(16 bits) Low
CL = CRC Low. Checksum according Modbus low nibble
CH = CRC High
1H = Data 1 High
1L = Data 1 Low
2H = Data 2 High
2L = Data 2 Low

Other parameters

```
Const Initializing = &HFE
```

The device will always react on this address. It is foreseen to make initial programming of device parameters and address: Myaddress.

```
Const Adressparamter = 10            ' Slaveparameters position
Const Parameters = 9                 ' Slaveparameters position
```

These pointers define the position in the data’s volatile RAM array

```
Const Dataarea = 16                  ' Databuffer + 2 Slaveparameters
```

This is the size of the data area.

```
Const Savetoeram = 10                 ' Address of Myaddress
```

This pointers are used to define the position in EERAM.

```
Const Maxchar = 19                   ' Nr. of characters for serial buffer
```

Size of the receive ring buffer

```
Const Telegrambytes = 14            ' Nr. of transmitted Bytes
```

Max length of the used telegrams. More important in the bigger devices like 8515.

```
Const Pause = 2                     ' Wait time before answer
```

A small device like this AVR reacts to fast for a SCADA system. With this parameter you adjust a small wait time for the answer telegram.

Pseudo constants programmed via telegram

```
Dim Myaddress As Eram Byte At &H02          ' Address for this node (2-253)
Dim Myparameter As Eram Byte At &H03         ' Address for this node (2-253)
```

Important Variables

```
Dim Bytebuffer(maxchar) As Byte             ' Receive serial-buffer
Dim Db(telegrambytes) As Byte               ' Data’s in telegram
Dim Sdb(telegrambytes) As Byte              ' Data’s in send telegram
Dim Datas(dataarea) As Byte At &H60         ' Place to keep the dynamic data’s
```

Define port parameters

Port B Is Used For Data Exchange With The Latches

```
'Ddrb = &HFF                     ' WRITE to multiple I/O Data port
'Ddrb = &H00                     ' READ from multiple I/O Data port
```

Port D is used for communication and for latch controll

```
Ddrd = &B01111100                   ' Set Portd Pin 2..6 as Outputs
‘ Ddrd.0                 ' RXD
' Ddrd.1                 ' TXD
' Ddrd.2                 ' Strobe Enable Read chip 1
' Ddrd.3                 ' Strobe Enable Read chip 2
' Ddrd.4                 ' Strobe output latches Allegro UCN 5801A  1
' Ddrd.5                 ' Strobe output latches Allegro UCN 5801A  1
' Ddrd.6                 ' Enable for RS422 send pin
' Ddrd.7                 ' NC for 2313
```

Setup interrupts

Allow receive & transmit ISR
Config Timer 0

Transfer from ERAM slave address and parameters

```
 Datas(adressparamter) = Myaddress        ' Set new Slave address
 Datas(parameters) = Myparameter         ' Set new Slave parameters
```

_com_z:

Startingpoint for telegrams
If received address is this node or the general initialization address, read next Byte from master, if not return to start

_com_1:

If next character is one of the three solved commands wait for next chars or restart

```
  Select Case Db(2)
        Case 3 :                   ' A read command
        Case 6 :                   ' A write command
        Case 16 :                  ' A write command
   Else com_z                     '  Wrong command = restart
   Else                       ' Wait for next incoming data
```

_com_2:

Get telegram into binary mode
Clear the receive buffer to make a calculated response possible
Check the CRC of the message for errors

_com_3:

No CRC errors in packet so check what to do and start reading / writing
Select Case of telegram

```
Case 03 :                         ' READING datas 1..n
   ' Request Read    Holding Register(03)   SS 03 SH SL PH PL CL CH
   ' Answer  Read 16 Holding Register(03)   SS 03 BC 1H 1L CL CH
   ' Answer  Read 32 Holding Register(03)   SS 03 BC 1H 1L 2H 2L x  x CL CH
   Gosub _send                   ' Give answer
   Goto _com_z                   ' All done, go back to Start
```

```
Case 06 :                        ' WRITING data’s 16 bit
       ' Request Preset Single Register(06)     SS 06 RH RL 1H 1L CL CH
       ' Answer Preset Single Register(06)      SS 06 RH RL 1H 1L CL CH
       Gosub _send                     ' Give answer
       Goto _com_z                     ' All done, go back to Start

Case 16 :                        ' WRITING data’s 1..n
  ' Request Preset Multiple Register(16) SS 10 SH SL PH PL BC 1H 1L 2H 2L CL CH
  '  Answer Preset Multiple Register(16)  SS 10 SH SL PH PL CL CH
      Gosub _send                      ' Give answer
      Goto _com_z                      ' All done, go back to Start
 End Select
Goto _com_z                              ' Not identified command
```

Subroutine for sending values

Start CRC generate routine and calculate CRC for all sending bytes
Send Packet To Master , Including The Sync Byte

Interrupt routine for preparing serial port

On powerup it's treated once !

Interrupt routine for reading serial input

Read into serial ring buffer

Interrupt routine for sendig serial output

Send contents from buffer

Interrupt routine for multiplexing In / Outputs

```
Timer0_isr:                           ' Controls the multiplexing of data versus I/O's
'   Port D is used for communication and for latch control
'   Ddrd.2                    ' Strobe Enable Read chip 1
'   Ddrd.3                    ' Strobe Enable Read chip 2
'   Ddrd.4                    ' Strobe output latches Allegro UCN 5801A  1
'   Ddrd.5                    ' Strobe output latches Allegro UCN 5801A  2
'   Ddrd.6                    ' Enable for RS485 send pin

Select Case
Case 0:                          ' READ 1 --- High Byte ---
Case 1:                          ' READ 2 --- Low Byte ---
Case 2:                          ' WRITE 3 --- High Byte ---
Case 3:                          ' WRITE 4 --- Low Byte ---
```

Additional features:

1. When slave address is FF means not yet programmed, a HW test is possible, because signal of input 1..16 is copied to output 1..16.

3. The second output register (bytes 07 and 08) has a combined function. When set output pin.x goes high, until the according input.x shows high. This feature is foreseen as fast “end-position-reached” digital loop, & is AND’ed with outputs!

2. The one byte parameter defines by AND function with input 1..8 counts up a counter (bytes 07 and 08) at each signal change.