ICS ELECTRONICS
division of Systems West Inc.

IEEE 488 APPLICATION BULLETIN

USING A 4894A or a 4804 TO CONTROL MODBUS DEVICES FROM THE GPIB BUS

INTRODUCTION

This application note shows how to use an ICS Model 4894A or a 4804 GPIB-to-Serial Interface to control Modbus Devices from the GPIB bus. This application note describes the Modbus protocol, the connections from a 4894A or 4804 to a Modbus device and the software that translates commands into the Modbus RTU protocol. This application note also includes an example Visual Basic program that lets a user control Watlow Controllers from a PC keyboard.

MODBUS PROTOCOL

Modicon Corporation, now AEG Schneider, created a serial protocol known as Modbus for controlling industrial process control systems. The Modbus protocol is an extremely reliable protocol for exchanging data in noisy industrial applications. The Modbus protocol is a packet exchange protocol where the controller receives a response to each packet that it sends to a device. Each packet contains the address of the controller that is to receive the information, a command field that says what to do with the data fields and a CRC (Cyclic Redundancy Checksum) field that is used to validate the packet. The Modbus protocol uses two coding schemes, ASCII or HEX characters. The HEX character format is known as the RTU protocol.

Figure 1 shows an Modbus HEX RTU message packet. The Address and Command fields are 8-bits. The Register-Data field holds multiple 16-bit words. The CRC field is a 16-bit word. The protocol uses asynchronous serial characters with single start and stop bits. Each character holds 8 bits or 2 HEX characters.

Packet Format:	Addr	Cmd	Registers.... .data	CRC

Figure 1 Modbus Packet

ICS's GPIB-to-SERIAL INTERFACES

ICS's GPIB-to-Serial Interfaces are IEEE-488.2 compatible interfaces with several features that make them ideal for interfacing the GPIB bus to a Modbus device. Both units operate at baud rates up to 38,400 baud and have RS-232 and RS-422/RS-485 interfaces for driving Modbus devices in a one-to-one basis or as a network controller. Both units transparently convert the binary data characters used in the Modbus RTU protocol between the GPIB bus and the serial communications link.

The Model 4894A GPIB-to-Serial Interface is housed in a small metal case and has an optional kit for mounting in a 19 inch instrument rack. The Model 4804 GPIB-to-Serial Interface is a 4.5 x 5.5 inch board that is intended to be built into a chassis or into a piece of test equipment.

Both units operate similarly and have the same control and data modes. At power turn-on, they power up in the data mode where they transparently transfer data between the GPIB bus and their serial interfaces. An escape sequence switches the units into a control mode where the serial settings can be set by commands from the GPIB bus. The settings are saved in internal nonvolatile memory.

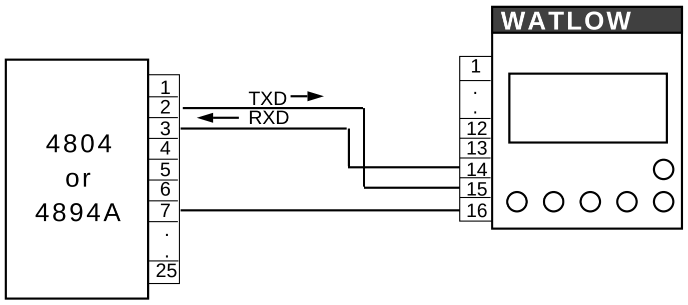


Figure 2 RS-232 Connections to Watlow Controller

CONNECTING TO WATLOW CONTROLLERS

All the new Watlow Process Controllers use the Modbus RTU protocol and are the example Modbus device for this Application Note. The F4 series Controllers have both RS-232 and RS-485 interfaces and can be easily connected to the 4894A or 4804 GPIB-to Serial Interface. The RS-232 signals can be used for short distances and require only three wires as shown in Figure 2.

RS-485 signals are best for distances over 100 feet, for electrically noisy environments, or for connecting several controllers together on a serial network. An RS-485 network uses two wires that are driven only when a device is transmitting. A resistor termination network holds the two signals in a mark condition when none of the devices on the network are transmitting. The termination network is a 1 Kohm pullup, a 120 to 200 ohm load resistor and a 1 Kohm pulldown resistor as shown in Figure 3. One termination network is sufficient for short networks. For networks over 100 feet long or in very noisy environments, place a separate termination network at each end of the cable. (For more information about RS-485 serial data transmission and termination networks, refer to ICS's 4894A/4804 Manual or visit our associated website at http://www.icsdatacom.com)

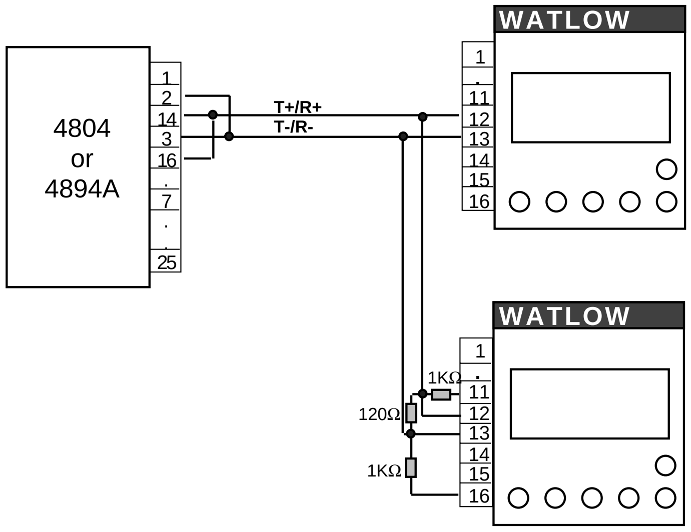


Figure 3 RS-485 Network Connection

In Figure 3, the termination network is placed on the last RS-485 device on the network, in this case a Watlow Controller. Terminal #11 on the Watlow F4 series controllers is a + 5 Vdc output terminal provided to power the termination network. Terminal #16 is the controller's ground pin and is used for one end of the pulldown resistor.

OVERALL SYSTEM

Figure 4 shows the overall system configuration. The PC is the system controller and the Watlow Process Controller is a device on the Modbus network. The transmission program in the PC converts user commands into messages that conform to the Modbus protocol. The PC then addresses the 4894A or 4804 to listen and sends it a packet message that is terminated with EOI asserted on the last character. The 4894A /4804 converts the GPIB message in to a serial packet and transmits it to the Watlow Controller. The Controller's response is received by the 4894A/4804 and talked out on the GPIB bus with EOI on the last byte when the unit is addressed to talk. The PC program can either wait for a preset amount of time before reading the response packet or the 4894A/4804 can be set to generate a service request (SRQ) when the response packet is received from the Watlow Process Controller.

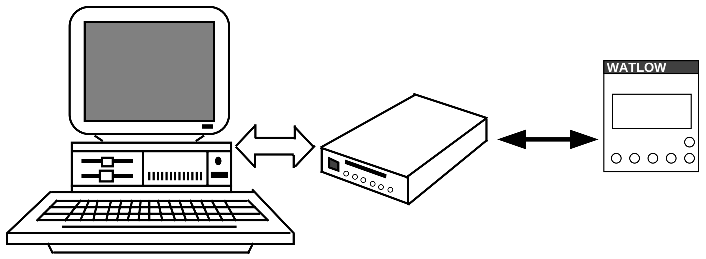
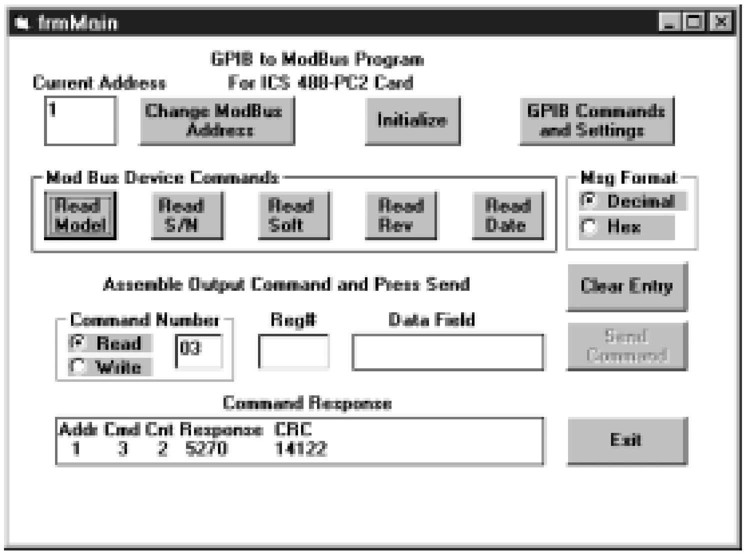


Figure 4 System Configuration

PROTOCOL CONVERSION SOFTWARE

ICS has developed a Visual Basic example program, GPIB2MOD, that shows how a 4804 or a 4894A can be used to communicate with a Modbus device from the GPIB bus. The program assembles device address, command number and command information into variables that are passed to a subroutine that prepares the Modbus packet. The packet is then outputted to the 4894A or 4804 which transmits it serially to the Modbus device. Received messages are checked by a subroutine for valid CRCs, then broken apart and the components are returned to the user. The GPIB2MOD program displays the results for the user. The subroutines do the bulk of the message preparation work and make it easy to adapt the example program to a real application.

The program has two forms, a Main form and a GPIB form. The Main Form lets the user set the Modbus device address, send specific queries to the Modbus device and a window for displaying the response. The Main Form also has boxes where the user can assemble custom commands by selecting read or write and entering the register number and register or data words. Clicking the SEND button, assemblies and outputs the message. Figure 5 shows the Main Form with the response to a Read Model# query.



Current Address: 1

Msg Format: [✓ Decimal] Hex

Command Number: [✓ Read] Write 03

Reg#:

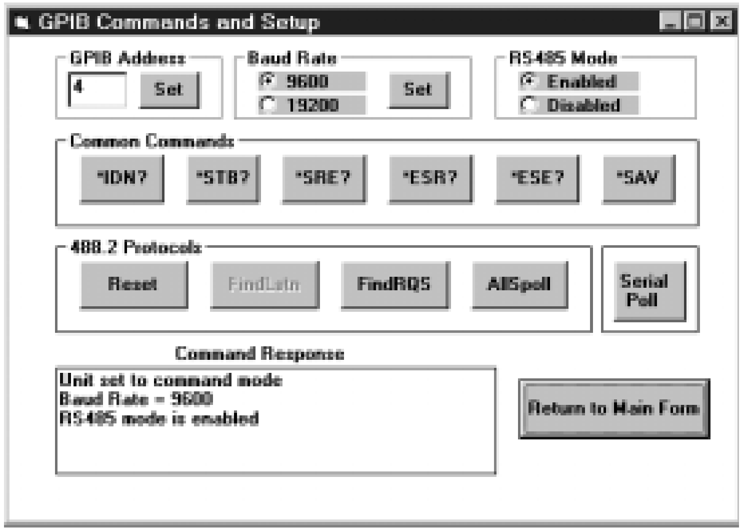
Data Field:

Command Response

```
Addr Cmd Cnt Response  CRC
  1    3   2   5270    14122
```

Figure 5 GPIB2MOD Main Form

The GPIB Form has numerous controls for setting and querying the status of the 4894A or 4804. The GPIB Form enters the GPIB address of the 4804 or 4894A into the program, sets the Baud Rate and enables or disables the RS-485 mode. The form's other controls query many of the 4894A or 4804's IEEE-488.2 Status Registers, execute several 488.2 protocols and serial poll the device. These are more controls than you probably want or need in your application program but they were included in the example to show the device's capability. Figure 6 shows the GPIB Form as it appears when first loaded with the device address set to 4, 9600 baud and RS-485 enabled.



GPIB Address: 4

Baud Rate: [✓ 9600] 19200

RS485 Mode: [✓ Enabled] Disabled

Command Response

```
Unit set to command mode
Baud Rate = 9600
RS485 mode is enabled
```

Figure 6 GPIB Form

The GPIB2MOD(bus) program has two .bas files. PC2W32.bas contains the GPIB constants, variables and function declarations for the GPIB calls. MODBUS.bas contains the variables, constants and subroutines that assemble and dissemble the Modbus packets.

The program communicates with the GPIB bus by making ICS style driver calls to a 488-PC2 card in the PC. The program was also compiled with a different PC2W32.BAS file that uses a translation DLL to make NI 488.2 style calls that work with ICS's PCI and PCMCIA GPIB Controller Cards or any GPIB Controller Cards that use the National Instruments 488.2 Commands. Copies of the program can be obtained from ICS's web site at http://www.icselect.com.

GPIB2MOD PROGRAM OPERATION

When the GPIB2MOD program is first started, only the INITIALIZE button is enabled on the Main Form. Clicking the INITIALIZE button initializes the GPIB Controller Card, the 4894A or 4804 and the 488.2 Driver software. If the initialization is successful, the remaining Main Form controls are enabled. The user should Click the GPIB COMMANDS AND SETTING button to go to bring up the GPIB Form. Verify that the GPIB address is the same as the address shown on the 4894A or 4804 LEDs at power turn-on. The program defaults to ICS's factory default GPIB address of 4. If the 4804 or 4894A is set to a different address, the address will have to be changed in the GPIB Address box. Press Set when done to update the program. Use the *IDN? button to query the 4894A or 4804's IDN message and verify communication with the GPIB interface and the correct address setting.

When the GPIB Form opens, the program queries the 4894A/4804 to determine its baud rate and RS485 mode. The answers are displayed in the Test box as shown in Figure 6. Verify that the baud rate setting matches the Watlow controller setting. If the controller is connected to the 4894A/4804's RS-485 terminals, then the RS485 Mode should be enabled. This sets the 4894A/4804 to tristate its transmitter when not transmitting data. Click the EXIT button to return to the Main Form.

On the Main Form, the Modbus device address defaults to 1 which should match the Watlow Controller's address. To change the address, enter a new value in the window and click the SET button. Click the READ MODEL button to read the controller's model number. The response displayed in the Command Response window is decoded to show the device address, command number, number of response bytes, the response data and the CRC number. The user can select a decimal or HEX coded display by clicking the option buttons in the MSG FORMAT frame.

The remaining buttons in the MODBUS DEVICE COMMANDS frame query the serial number, software revision, hardware revision, and date settings in the Watlow controller.

The user can send all but the most complex commands to the Watlow Controller or any Modbus device by filling in the command windows under the ASSEMBLE OUTPUT COMMAND legend and clicking SEND. Use the Read or Write options to select a command number or enter a value in the COMMAND NUMBER window. Enter the register number in the REG# window. Enter any data to be sent in the Data Field. Multiple data entries must be separated by commas. The SEND button is enabled when the COMMAND and REG# fields are filled in. The CLEAR ENTRY button clears the COMMAND, REG# and DATA windows for a new command.

WRITING A CUSTOM PROGRAM

The user can easily adapt the GPIB2MOD program's .bas component files to any Visual Basic program to control Modbus devices via the GPIB bus. The cmdSEND (click) listing in Figure 7 shows how to call the *build_modbus_message* function which assembles the Modbus packet. The calling function needs four parameters: command type, Modbus device address, starting register number and data values. In the GPIB2MOD program, the command type, starting register and Modbus device address are entries in text boxes. The data values are an array of up to 40 entries terminated with the END_OF_DATA_VALUES constant. For Watlow, each entry is a 16 bit value and depends upon the command being sent to the controller.. When done, call ICS's *ieOutputB* function to send the packet (Outstring$) to the 4894A or 4804. If you are using a GPIB output command for another card or command language, select the termination option that only asserts EOI on the last character.

Provide a short delay (50 ms minimum) to give the Watlow controller time to receive the serial message and send a response back to the GPIB interface, before reading the response message. The messages tested in this program only took 7 ms, but you need to have some margin for more complex messages. An alternative method would be to have the 4894A or 4804 generate a Service Request when it had data in its RX buffer and then provide a short delay of 10 ms to allow the remainder of the message to be received before inputting the response packet. Caution - Do not make the input call until the complete response message has been received by the 4804 or 4894A or else you can loose part of the response.

Follow the read back part of the listing to read the Modbus response. Use the *receive* function in the Main Form to read back the response message. The *receive* function uses ICS's *ieEnterB* function to read in the response message, discards any excess space characters and returns the response message. If you are using another GPIB input command, use a termination option that only terminates when EOI is asserted. The Msg_Format flag is set by the user clicking HEX or Decimal values. Based on the Msg_Format flag setting, call either the *convert_modbus_data_to_decimal* function or the *convert_modbus_data_to_hexidecimal* function to dissemble the response message into its components and recover the response data. The GPIB2MOD program displays the complete response, but you can discard any unnecessary information.

If the user's program is written in a Basic language, the Visual Basic statements are easy to convert. If the user's program is in C, the Visual Basic example will serve as a guide to the new program.

SUMMARY

This application note has described how to drive a Modbus device from the GPIB bus using ICS's Model 4894A or 4804 GPIB-to-Serial Interfaces. Watlow's new F4 series of Process Controllers were used as the example device. The wiring examples and sample code have been tested with a Watlow F4 series controller and should be easily adapted to other Modbus devices.

Main Form cmdSEND_Click listing

```
Private Sub cmdSend_Click()  ‘sends general message
 Static data_value!(40)
 comdtype% = Val(txtCmdNum.Text)   'command type
 Start_reg! = Val(TxtReg.Text)          'starting register

 If Len(txtOutput.Text) > 0 Then
   Data$ = txtOutput.Text
   If InStr(txtOutput, “,”) = 0 Then
     data_value!(1) = Val(Data$)            'data values
     data_value!(2) = END_OF_DATA_VALUES
   Else
     I = 1
     Do
        L = InStr(Data$, “,”)
        If L = 0 Then
          newstr$ = Left$(Data$, (Len(Data$)))
          data_value!(I) = Val(newstr$)
        Else
         newstr$ = Left$(Data$, (L - 1))
         data_value!(I) = Val(newstr$)
         newdatastr$ = Right$(Data$, (Len(Data$) - L))
         Data$ = newdatastr$
        End If
        I = I + 1
      Loop Until L = 0
     data_value!(I) = END_OF_DATA_VALUES
   End If

 Else
   data_value!(1) = 1
   data_value!(2) = END_OF_DATA_VALUES
 End If
OutString$ = build_modbus_message$(comdtype%,
MBDevice%, Start_reg!, data_value!())
 L = Len(OutString$)
 ioerr% = ieOutputB(CardDev, OutString$, L)
                                              ‘output command
 ProcessError (ioerr%)  'checks  for GPIB command errors

‘readback
  Call Tdelay(0.3)                ‘delay for serial data
  Instring$ = Receive_msg$()     'input response
  RxLen% = Len(Instring$)
  ProcessError (ioerr%)
  message$ = Instring$
  ‘Msg_Format = DECIMAL_DATA_FMT
                                  'flag set by panel control

 Select Case Left$(message$, 3)
   Case “Bad”
   ‘something was wrong with the message data.
       response$ = message$
       read_data_reg!(1) = -99999
   Case Else
     response$ = Instring$
     Select Case Msg_Format
       Case DECIMAL_DATA_FMT
        ‘convert the modbus response to decimal numbers.
            response$=convert_modbus_data_to_decimal$
                      (MESSAGE_RECEIVED, response$)
       Case HEX_DATA_FMT
      ‘convert the modbus response to hexadecimal numbers.
            response$ = convert_modbus_data_to_hex$
                      (MESSAGE_RECEIVED, response$)
     End Select
    txtResults.Text = Explaination$ & NL & “  “ & response$
  End Select
  txtDataEntry.Visible = False
End Sub
```

Figure 7   Main Form cmdSEND_Click listing