The Nuts and Volts of BASIC Stamps

Parallax, Inc.
www.parallaxinc.com

Nuts and Volts
www.nutsvolts.com

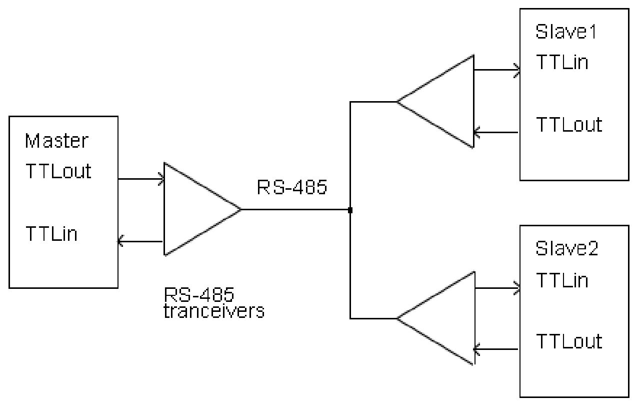
Column #56, December 1999 by Lon Glazner:

Stamp Net Part 2 – A Multi-drop Stamp-based Network

Last month, we designed an RS-485 based network, and de-scribed a communication protocol, which allowed multiple BASIC Stamp2-SXs (BS2-SX) to communicate together. The communication was a Master-Slave format in which the Master would send a communication string to a specific Slave, causing that Slave to execute a specific program stored in memory. Once the program was executed, the Slave would respond with some data. This network was called STAMP Net.

Our previous design and testing was done on a solderless breadboard, which allowed enough flexibility to prove out the design concept. This month, the STAMP Net design was moved to a set of prototype PCBs, and the network communication was tested with cables measuring roughly 200 feet. From the literature I’ve read on RS-485, a 200-foot cable is small potatoes, as far as cable lengths go. And judging from the lack of trouble that I had getting the system to work, I’d have to agree.

Figure 56.1: Simplified RS-485 network



Defining the Design

This design was defined pretty well in last month's article. I did decide to add a little code to control a 1A relay, and a pair of general-purpose I/O pins. Even with the addition of this functionality, there is plenty of room left in the BS2-SX to implement more functionality. I would really like to add a PC terminal interface to the design, but that will have to be done some time in the future, when time permits.

As a little refresher from last month, I've included the system block diagram. Figure 56.1 gives a pretty good conceptual overview how a Master-Slave communication protocol on an RS-485 network is connected. The RS-485 transceivers that were chosen for the design are the MAX487 from Maxim Integrated Circuits.

The changes in the schematic were pretty simple. A relay driver circuit was added. The relay is a 5V variety (drive voltage) which can handle 1A with a 30V potential at its switch connections. A 2N3904 NPN transistor buffers the BS2-SX pin that drives the relay, and a 1N4001 diode was added across the relay's inductor to reduce voltage spikes and other transients.

Two general-purpose I/O pins (GPIO, for short) were included, and should have added protection if they are interfacing to external devices. Take a look back at the Sept. '99 Stamp Applications article for more information on protecting I/O pins.

The GPIO pins can be set as inputs or outputs in software. The voltage level at either GPIO1 or GPIO2 will be returned to the Master unit if either pin is set as an input pin. As inputs, these pins could be used to read switches at each STAMP Net node. These same pins could be configured as outputs by a Master unit and be used to drive additional relays, or be used to light LEDs.

Table 56.1: Command and response string definition

Byte	Number	Description
Address	1	Address of unit message is intended for
Program	2	Program to be executed by receiving unit
Data 1	3	General purpose data byte
Data 2	4	General purpose data byte
Data 3	5	General purpose data byte
Data 4	6	General purpose data byte
Data 5	7	General purpose data byte
Checksum	8	Sum of all bytes in message

Figure 56.2: Schematic of each STAMP net node

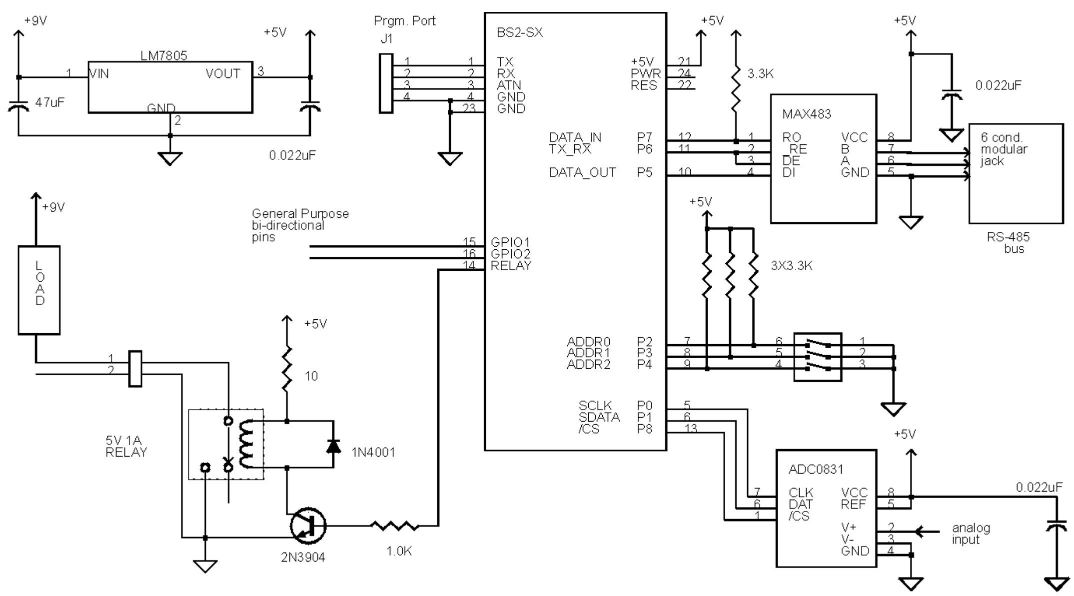


Figure 56.3: Software response timing requirement

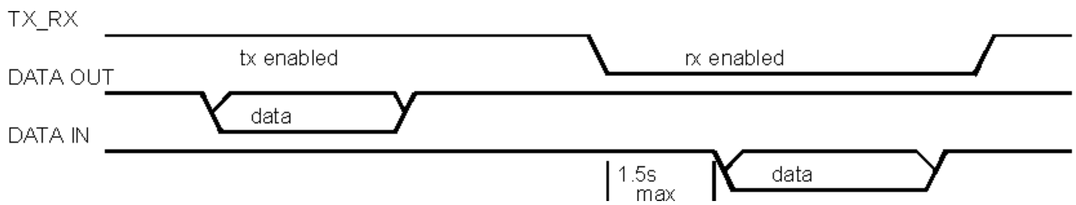


Figure 56.4: A master and slave transaction

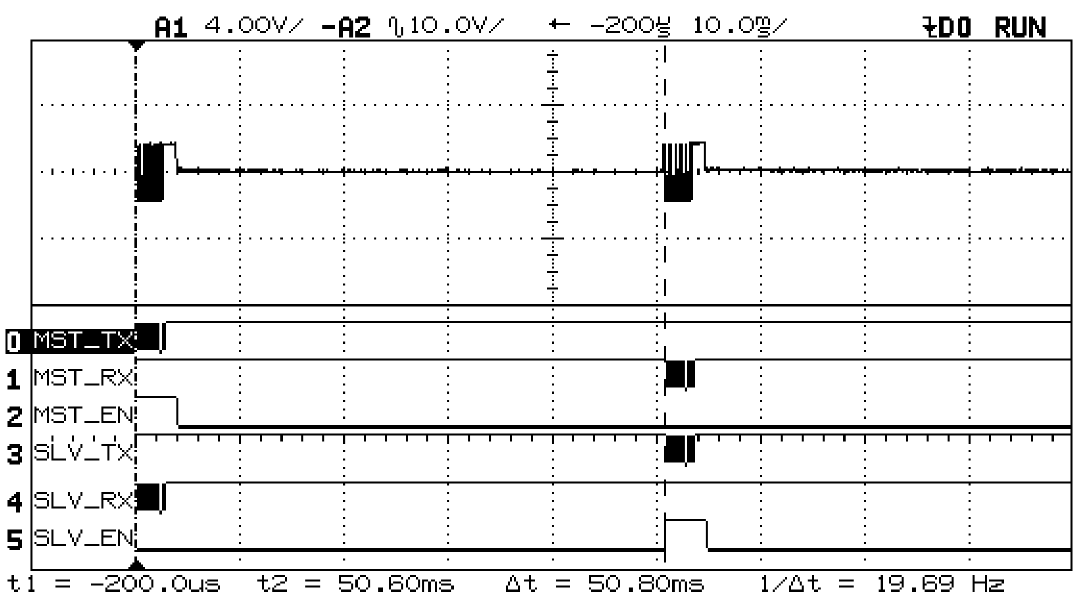


Figure 56.5: A zoom on the master unit transmission

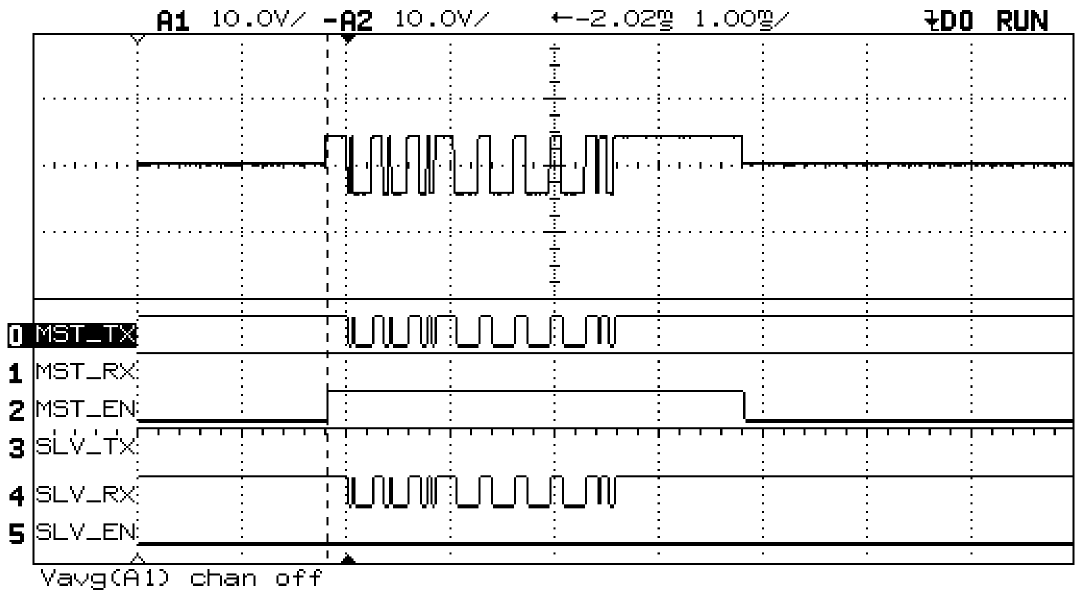
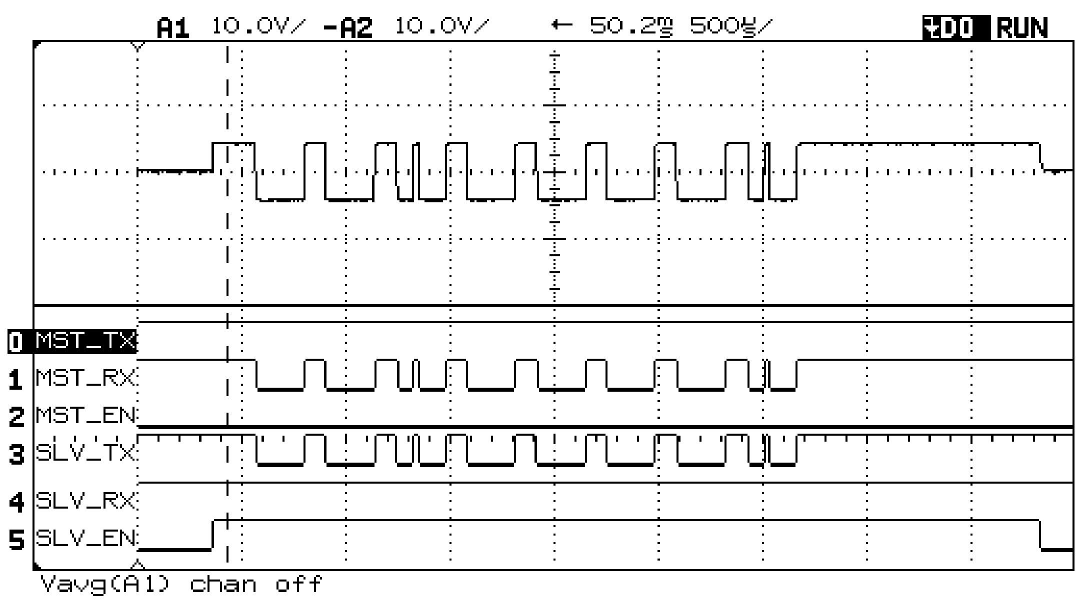


Figure 56.6: The slave unit response



Reviewing Our Communication Protocol

The communication protocol was based on a generic structure of eight bytes for each communication string. The Master unit sends the address of the STAMP Net node that it was trying to communicate with, and the number of the program that it wishes to have executed. Five generic data bytes and a checksum byte follow the first two bytes. Table 56.1 can be used as a quick reference for the data string format for STAMP Net.

As of now, there are only two programs that are loaded into the STAMP Net nodes for the Master unit to access through a Slave unit.

The first is the Analog.bsx program, which reads an eight-bit A/D and returns the average, maximum, and minimum measurements out of 128 samples. For the Master unit to receive the A/D data, it only needs to send a valid communication string with the Program byte set to 1. The Slave unit will interface to the A/D, take 128 samples, and return the results in the Data 1, Data 2, and Data 3 positions, and then return to waiting for data from the RS-485 bus.

The IOControl.bsx program operates a little bit differently than the Analog.bsx program. The value in Data 1 — when it is received from a Master unit that requests Program 2 — is used to determine the direction (input or output) of the GPIO pins, as well as the output voltages (0V or 5V) of the relay pin and any GPIO pins set as outputs.

These two programs are prime examples of the kinds of control that your Master unit can have over a Slave unit. The Master may execute a program in a Slave, and that program can be performed more or less autonomously. Or the Master may send data along with the program request information that reconfigures the Slave unit as in the IOControl.bsx program.

A communication protocol can have multiple timing requirements. I left the timing requirements for STAMP Net very lax. In fact, I extended the response time allotted for the Slave unit from 500ms to 1.5s since last month's article. I never intended to make STAMP Net a high-speed network, and some of the STAMP Net nodes that I'll be using will be monitoring environmental conditions, so additional time might be necessary.

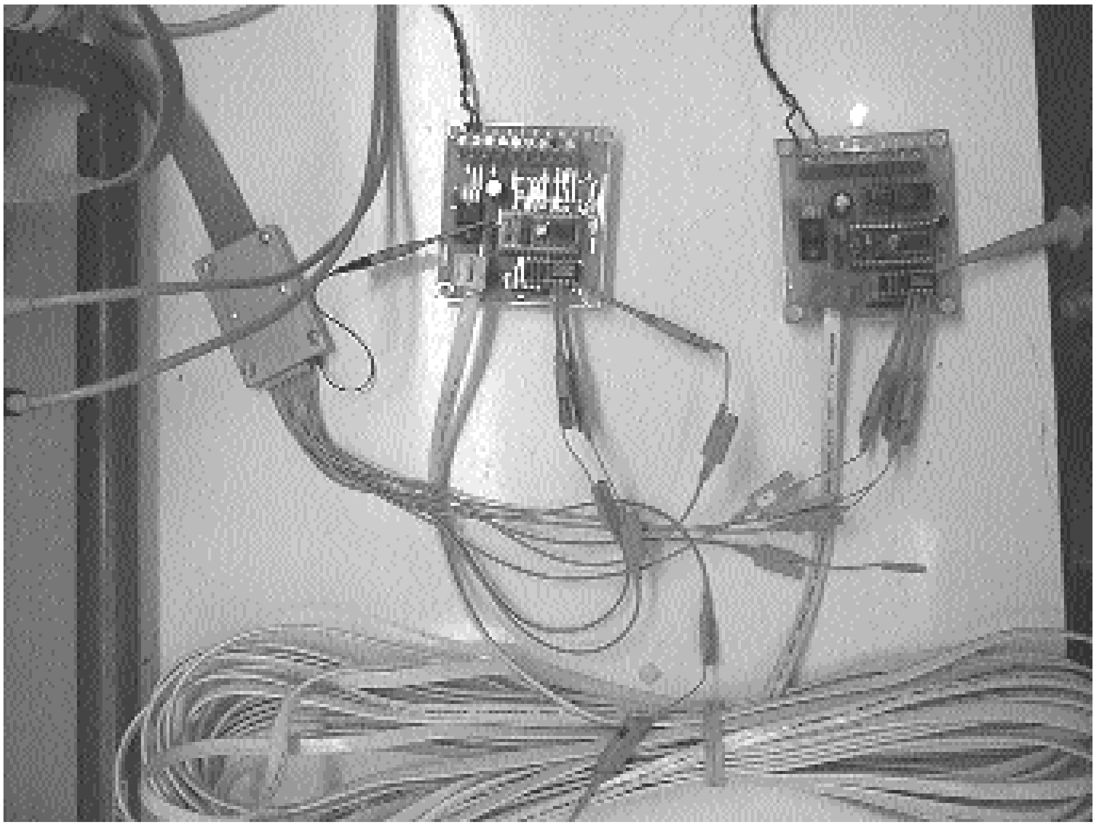
So what does this all look like electrically? Oscilloscope captures of the Master and Slave communication are shown in Figure 56.4. Take a look at the data on the Master TX line (MST_TX), and how it is received on the Slave RX line (SLV_RX). The receive enable line is also of interest in this oscilloscope capture (MST_EN and SLV_EN). It is low to set the MAX487 IC to receive data, and high for the MAX487 to send data to the RS-485 bus (keep in mind that both the Master and Slave control their own MAX487). If you're having trouble with a communication protocol, "scoping" the signals like I've done here can detect the most common problems. The top signal, without a label, is scope channel A1–A2, which is the differential voltage as seen by the MAX487.

Zooming in on the Master unit transmission gives us some additional insight into what's really going on. It is also nice to get a close-up of the Slave unit response, as shown in Figure 56.6.

The Software

There are a few differences between the Master.bsx and Analog.bsx programs from last month. I added a couple of lines of code which allowed me to skip the I/O direction setting lines of the code if the IOControl.bsx program had been executed. This prevents the Slave unit from resetting the direction of the GPIO pins, or their output voltage levels when the Slave returns to the main program after IOControl.bsx is executed.
I also removed the part of the Master.bsx routine which polled each Slave on the STAMP Net. This function will be replaced later by a user interface, and a more in-depth polling routine. I have added the new versions of the programs here for anyone that may wish to see them. For these programs, the Slave address and Program request byte should be entered into the Master part of the Master.bsx program. This makes testing new programs a little simpler.

Figure 56.7: STAMP Net Under Test



In Closing

At this point, I've got a pretty good head of steam going on the STAMP Net project. The PCBs are designed and look like they'll work for the final application, which is a workshop alarm. Although, I've got to say, there is still a lot of software to write. The three programs listed here need to be refined, and I'm sure I've got a little learning to do. But, with the BS2-SX, I have plenty of room for code, and the Parallax technical support is only an E-Mail away.

The RS-485 network went together without much trouble, and termination resistors were avoided (thanks to short cable requirements). This is not always the case. I think I stacked the deck in my favor by taking a realistic approach to what I wanted my network to do. The generic STAMP Net communication protocol prevented me from getting bogged down in timing issues, or overly elaborate code requirements. Keeping cables short (no run over 200 feet long) kept expenses down, and electrical reflections under control.

Probably the best news is that I've got about 80% of the BS2-SX resources still available for use. That could translate into a pretty cool user interface. We'll just have to wait and see.

But that's it for STAMP Net in this forum. I hope everyone found something of interest or at least something useful for your own Stamp project. Next month, we'll take a step back to the past with a BASIC Stamp 1 and a 12-bit A/D interface. See you then.

```
'Program Listing 56.1: Master.bsx
'*****************************************************************
'Master Program
'The Master program controls communication and data display. For a
'unit designated as a Master unit(addresss = 0) this program is
'used to poll the various slave units. If a unit is a Slave unit
'(address <> 0) then this program is where the unit waits for
'commands.
'
'{$STAMP BS2SX,C:\Parallax\Analog.bsx,C:\Parallax\IOControl.bsx}
'0:Master_Prgm.bsx

'I/O pin designations
AD_Clk          CON     0               'ADC0831 clock pin
AD_Dat          CON     1               'ADC0831 data pin
AD_CS           CON     8               'ADC0831 chip select (asserted low)
Relay           CON     9               'Relay control pin (asserted high)
GPIO1           CON     10              'General purpose I/O pin
GPIO2           CON     11              'General purpose I/O pin

'Communication Constants
Data_Out        CON     5               'TTL data out pin
TX_RX           CON     6               'Receive enable (asserted low)
Data_In         CON     7               'TTL data in pin
Baud            CON     45              '38.4kbps, 8N1 true data

'Internally used registers
Addr            var     byte            'Address of unit
Comm_Flag       var     byte            'flag bits for unit
Mstr    var     Comm_Flag.bit0          'Set for Master unit clear for Slave
Sl      var     Comm_Flag.bit1
        'Set if Slave # 1 is present on RS-485 bus
S2      var     Comm_Flag.bit2 'Set if Slave # 2 is present on RS-485 bus
S3      var     Comm_Flag.bit3 'Set if Slave # 3 is present on RS-485 bus
S4      var     Comm_Flag.bit4 'Set if Slave # 4 is present on RS-485 bus
S5      var     Comm_Flag.bit5 'Set if Slave # 5 is present on RS-485 bus
```

```
S6      var     Comm_Flag.bit6 'Set if Slave # 6 is present on RS-485 bus
S7      var     Comm_Flag.bit7 'Set if Slave # 7 is present on RS-485 bus

'Communication message string variables bytes(8 total)
Addr_Req       var    byte           'Unit address of message destination
Prgm_Req       var    byte           'Request execution of this program
Dat1           var    byte           'Data byte 1
Dat2           var    byte           'Data byte 2
Dat3           var    byte           'Data byte 3
Dat4           var    byte           'Data byte 4
Dat5           var    byte           'Data byte 5
Checksum       var    byte           'Sum of previous bytes

'Storage Registers
Put_Addr       var    byte           'Put address location
Get_Addr       var    byte           'Get address location

'Working registers
Loop1          var    byte           'For...Next variable
Work1          var    byte           'General purpose register
Work2          var    byte           'General purpose register
Work3          var    byte           'General purpose register
Work4          var    byte           'General purpose register
WorkBig        var    word           'Word sized general purpose register

'A/D registers
ResultA_D      var    byte           'Result of A to D measurement
MaxA_D         var    byte           'Storage for maximum A to D result
MinA_D         var    byte           'Storage for minimum A to D result
AvgA_D         var    byte           'Storage for avg. A to D result

'Program constants
AD_Samples     CON    128            'Number of samples taken
DirFlag        CON    6              'Flag set to skip direction setting
routine

'*******************************************************************
Main_Program:

GET    DirFlag,Work2
If Work2 = 10 Then Get_Address

Comm_Flag      = %00000000
Outs           = %0000000100100000   'Set output pin values
Dirs           = %0000001101100011   'Set pin direction values

Get_Address:
Addr   = (INL&%0011100)/4            'Get unit address from P4-2
If Addr <> 0 then No_Master
       Mstr = 1
No_Master:
```

```
'Pause         2000

'Addr and Comm Flag register Debug statements
'Debug "Address = ", BIN8 Addr,CR
'Debug "Comm Flag = ", BIN8 Comm_Flag,CR

If Mstr = 1 then Master_Program
       Goto    Slave_Program

'********************************************************************
Master_Program:

Pause 1000
Addr_Req = 1                  'Contact unit 1
Prgm_Req = 2                  'Request IOControl Program
Dat1 = %11100100              'GPIO1 = out-high GPIO2 = input Relay = on

Checksum = Addr_Req+Prgm_Req+Dat1+Dat2+Dat3+Dat4+Dat5
HIGH           Data_Out       'Set output high
HIGH           TX_RX          'Enable transmission on RS-485
SEROUT Data_Out,Baud,[Addr_Req,Prgm_Req,Dat1,Dat2,Dat3,Dat4,Dat5,Checksum]
PAUSE          1
LOW            TX_RX          'Enable receiver on RS-485
SERIN
Data_In,Baud,1500,No_Data,[Work1,Work2,Dat1,Dat2,Dat3,Dat4,Dat5,Checksum]

                              'Test checksum
Work4   = Work1+Work2+Dat1+Dat2+Dat3+Dat4+Dat5
If Work4 <> Checksum Then Bad_Data
                              'Set flag for unit that responds
       Comm_Flag = 1
       Work3 = %00000001      'Set up pointer bit
       Work3 = Work3 << Addr_Req     'Rotate "1" into Slave location
       Comm_Flag      = Comm_Flag+Work3
       'Add pointer bit to to designate active Slave
                              'Display incoming data
       Debug "Address of Sender = ",DEC Addr_Req,CR
       Debug "Data byte 1    = ",DEC Dat1,CR
       Debug "Data byte 2    = ",DEC Dat2,CR
       Debug "Data byte 3    = ",DEC Dat3,CR
       Debug "Data byte 4    = ",DEC Dat4,CR
       Debug "Data byte 5    = ",DEC Dat5,CR

       Goto    Done_Polling
Bad_Data:
       Debug   "Checksum Invalid Addr: ",DEC Addr_Req,cr
       Goto    Master_Program
No_Data:
       Debug   "No Data Returned Addr: ",DEC Addr_Req,cr
       Goto    Master_Program
Done_Polling:
```

```
       Debug   "Comm_Flag = ", BIN8 Comm_Flag,CR
'      Pause          3000
       Goto    Master_Program


'*******************************************************************
Slave_Program:
       Debug   "Slave Program ",CR
       LOW            TX_RX          'Enable receiver on RS-485
       SERIN
       Data_In,Baud,[Addr_Req,Prgm_Req,Dat1,Dat2,Dat3,Dat4,Dat5,Checksum]
                                     'Test checksum
       If Addr_Req <> Addr Then Bad_Address
       Work4   = Addr_Req+Prgm_Req+Dat1+Dat2+Dat3+Dat4+Dat5
       If Work4 <> Checksum Then Bad_Sum
       PUT            0,Dat1         'Store data for other programs
       PUT            1,Dat2
       PUT            2,Dat3
       PUT            3,Dat4
       PUT            4,Dat5
       RUN            Prgm_Req       'Execute requested program
Bad_Sum:
       Debug   "Checksum Invalid: ",cr
       Goto    Slave_Program
Bad_Address:
       Debug   "Wrong Address: ",DEC Addr_Req,cr
       Goto    Slave_Program

       Goto    Get_Address
END
```

```
' Program Listing 56.2: Analog.bsx

'Analog.bsx
'This is program 1 for the STAMP Net design. If this program
'is requested then 128 analog measurements are taken with the
'ADC0831 analog to digital converter. The maximum, minimum,
'and average result are returned to the Master unit.

'I/O pin designations
AD_Clk         CON    0              'ADC0831 clock pin
AD_Dat         CON    1              'ADC0831 data pin
AD_CS          CON    8              'ADC0831 chip select (asserted low)
Relay          CON    9              'Relay control pin (asserted high)
GPIO1          CON    10             'General purpose I/O pin
GPIO2          CON    11             'General purpose I/O pin

'Communication Constants
Data_Out       CON    5              'TTL data out pin
TX_RX          CON    6              'Receive enable(asserted low)
Data_In        CON    7              'TTL data in pin
Baud           CON    45             '38.4kbps, 8N1 true data

'Internally used registers
Addr           var    byte           'Address of unit
Comm_Flag      var    byte           'flag bits for unit
       Mstr   var    Comm_Flag.bit0
'Set for Master unit cleared for Slave
       Sl     var    Comm_Flag.bit1
'Set if Slave # 1 is present on RS-485 bus
       S2     var    Comm_Flag.bit2
'Set if Slave # 2 is present on RS-485 bus
       S3     var    Comm_Flag.bit3 'Set if Slave # 3 is present on RS-
485 bus
       S4     var    Comm_Flag.bit4
'Set if Slave # 4 is present on RS-485 bus
       S5     var    Comm_Flag.bit5
'Set if Slave # 5 is present on RS-485 bus
       S6     var    Comm_Flag.bit6
'Set if Slave # 6 is present on RS-485 bus
       S7     var    Comm_Flag.bit7
'Set if Slave # 7 is present on RS-485 bus

'Communication message string variables bytes(8 total)
Addr_Req       var    byte           'Unit address of message destination
Prgm_Req       var    byte           'Request execution of this program
Dat1           var    byte           'Data byte 1
Dat2           var    byte           'Data byte 2
Dat3           var    byte           'Data byte 3
Dat4           var    byte           'Data byte 4
Dat5           var    byte           'Data byte 5
```

```
Checksum       var    byte           'Sum of previous bytes

'Storage Registers
Put_Addr       var    byte           'Put address location
Get_Addr       var    byte           'Get address location

'Working registers
Loop1          var    byte           'For...Next variable
Work1          var    byte           'General purpose register
Work2          var    byte           'General purpose register
Work3          var    byte           'General purpose register
Work4          var    byte           'General purpose register
WorkBig        var    word           'Word sized general purpose register

'A/D registers
ResultA_D      var    byte           'Result of A to D measurement
MaxA_D         var    byte           'Storage for maximum A to D result
MinA_D         var    byte           'Storage for minimum A to D result
AvgA_D         var    byte           'Storage for avg. A to D result

'Program constants
AD_Samples     CON    128            'Number of samples taken
DirFlag        CON    6              'Flag set to skip direction setting
routine

'*******************************************************************
Main_Program:

GET    DirFlag,Work2
If Work2 = 10 Then Get_Address

Comm_Flag      = %00000000
Outs           = %0000000100100000   'Set output pin values
Dirs           = %0000001101100011   'Set pin direction values

Get_Address:
Addr   = (INL&%0011100)/4            'Get unit address from P4-2

       WorkBig       = 0             'Clear average storage register
       MinA_D = 255                  'Set minimum to max output
       MaxA_D = 0                    'Set maximum to min output

Measure_Analog:
       For Loop1 = 1 to AD_Samples
              LOW           AD_CS
              PULSOUT AD_Clk,10
              SHIFTIN AD_Dat,AD_Clk,msbpost,[ResultA_D]
              HIGH          AD_CS
              WorkBig = Workbig + ResultA_D
              If ResultA_D < MaxA_D Then Test_Min
                     MaxA_D = ResultA_D
```

```
Test_Min:
               If ResultA_D > MinA_D Then Keep_Sampling
               MinA_D = ResultA_D
Keep_Sampling:
       Next
       AvgA_D = WorkBig/AD_Samples

Debug  "Average Storage = ",DEC WorkBig,cr
Debug  "Minimum A to D = ",DEC MinA_D,cr
Debug  "Maximum A to D = ",DEC MaxA_D,cr

       Checksum       = MaxA_D+MinA_D+AvgA_D
       HIGH           Data_Out              'Set output high
       HIGH           TX_RX          'Enable transmission on RS-485
       SEROUT
       Data_Out,Baud,[$00,$00,MaxA_D,MinA_D,AvgA_D,$00,$00,Checksum]
       PAUSE          1
       LOW            TX_RX          'Enable receiver on RS-485

       RUN     0                     'Return to main program
END
```

```
'Program Listing 56.3: IOControl.bsx

'IOControl
'The IOControl program sets the pin direction and output levels of the
'Relay,'GPIO1, and GPIO2 pins. The pin direction and voltage level are
'determined by the value in Dat1 sent by the Master unit.
'
'The values are defined as,
'      Dat1,2 GPIO1 direction 1 = output
'      Dat1,3 GPIO2 direction 1 = output
'      Dat1,5 Relay voltage level
'      Dat1,6 GPIO1 voltage level
'      Dat1,7 GPIO2 voltage level

'I/O pin designations
AD_Clk         CON    0              'ADC0831 clock pin
AD_Dat         CON    1              'ADC0831 data pin
AD_CS          CON    8              'ADC0831 chip select (asserted low)
Relay          CON    9              'Relay control pin (asserted high)
GPIO1          CON    10             'General purpose I/O pin
GPIO2          CON    11             'General purpose I/O pin

'Communication Constants
Data_Out       CON    5              'TTL data out pin
TX_RX          CON    6              'Receive enable(asserted low)
Data_In        CON    7              'TTL data in pin
Baud           CON    45             '38.4kbps, 8N1 true data
```

```
'Internally used registers
Addr           var    byte          'Address of unit
Comm_Flag      var    byte          'flag bits for unit
Mstr   var     Comm_Flag.bit0 'Set for Master unit cleared for Slave
Sl     var     Comm_Flag.bit1 'Set if Slave # 1 is present on RS-485 bus
S2     var     Comm_Flag.bit2 'Set if Slave # 2 is present on RS-485 bus
S3     var     Comm_Flag.bit3 'Set if Slave # 3 is present on RS-485 bus
S4     var     Comm_Flag.bit4 'Set if Slave # 4 is present on RS-485 bus
S5     var     Comm_Flag.bit5 'Set if Slave # 5 is present on RS-485 bus
S6     var     Comm_Flag.bit6 'Set if Slave # 6 is present on RS-485 bus
S7     var     Comm_Flag.bit7 'Set if Slave # 7 is present on RS-485 bus

'Communication message string variables bytes(8 total)
Addr_Req       var    byte          'Unit address of message destination
Prgm_Req       var    byte          'Request execution of this program
Dat1           var    byte          'Data byte 1
Dat2           var    byte          'Data byte 2
Dat3           var    byte          'Data byte 3
Dat4           var    byte          'Data byte 4
Dat5           var    byte          'Data byte 5
Checksum       var    byte          'Sum of previous bytes

'Storage Registers
Put_Addr              var    byte   'Put address location
Get_Addr              var    byte   'Get address location

'Working registers
Loop1          var    byte          'For...Next variable
Work1          var    byte          'General purpose register
Work2          var    byte          'General purpose register
Work3          var    byte          'General purpose register
Work4          var    byte          'General purpose register
WorkBig        var    word          'Word sized general purpose register

'A/D registers
ResultA_D      var    byte          'Result of A to D measurement
MaxA_D         var    byte          'Storage for maximum A to D result
MinA_D         var    byte          'Storage for minimum A to D result
AvgA_D         var    byte          'Storage for avg. A to D result

'Program constants
AD_Samples     CON    128           'Number of samples taken
DirFlag        CON    6             'Flag set to skip direction setting
routine

'*******************************************************************
Main_Program:

GET    DirFlag,Work2
If Work2 = 10 Then Get_Address
```

```
Comm_Flag       = %00000000
Outs            = %0000000100100000   'Set output pin values
Dirs            = %0000001101100011   'Set pin direction values

Get_Address:
Addr    = (INL&%0011100)/4            'Get unit address from P4-2

'Set I/O Direction
       GET            0,Work1         'Dat1 from Mater stored in RAM0
       Work1.NIB0     = Work1.NIB0&%1100
'Clear 2 low bits of desired dir. reg.
       Work2.NIB0     = DIRC&%0011
'Clear 2 high bits of actual dir. reg.
       DIRC             = Work2.NIB0+Work1.NIB0
'Sum of high and low bits is direction
       DEBUG   "DIRC   = ",BIN4 DIRC,CR       'Display direction nibble

'Set output levels
       GET            0,Work1         'Dat1 from Master stored in RAM0
       Work1.NIB1     = Work1.NIB1&%1110
'High nibble has voltage levels
       OUTC           = Work1.NIB1    'Set output levels
       DEBUG   "OUTC   = ",BIN4 OUTC,CR       'Display output register

       Dat1           = 0             'Clear returned data
       If DIRC = %1111 Then No_Inputs
'If direction = %1111 then skip
               Dat1    = INC&%1100    'Dat1 equals values at GPIO1,2
       DEBUG   "INC    = ",BIN4 INC,CR 'Display input values

No_Inputs:                            'Send data to Master
       Checksum       = Dat1
       HIGH           Data_Out        'Set output high
       HIGH           TX_RX           'Enable transmission on RS-485
       SEROUT  Data_Out,Baud,[$00,$00,Dat1,$00,$00,$00,$00,Checksum]
       PAUSE          1
       LOW            TX_RX           'Enable receiver on RS-485

       PUT            DirFlag,10
       RUN            0               'Return to main program
END
```