Column #55, November 1999 by Lon Glazner:

Stamp Net Part 1 – A Multi-drop Stamp-based Network

There's been a lot of talk these last few years about the Internet and the creation of an information super-highway here in America. With all of the attention paid to the phenomena of the world wide web, I thought it was high time to bring the BASIC Stamp into this arena of networked electronics.

Think of STAMP Net as your very own information super-"dirt road." Okay, so it's not as cool or revolutionary as the Internet. You won't be able to exchange jokes with your co-workers when you're supposed to be slaving away for the boss. In fact, it'll be pretty slow and somewhat application specific. But for those of you who have considered automating your house with BASIC Stamps, or centralizing the control of some manufacturing equipment, STAMP Net should get you on your way.

The STAMP Net design consists of both the hardware communication architecture, as well as a software communication protocol required to connect a group of BASIC Stamp2 SXs (BS2-SX) together over an RS-485 network.

Defining the Design

We have a unique opportunity with this design. Typical engineering revolves around problem solving. Parts are selected, and software is written to solve a specific problem with time-to-develop and system cost being considered. STAMP Net is not such a design. Instead, it could be considered a design looking for an application. It could be used as a greenhouse monitoring system, a home lighting control network, or maybe as the backbone of a workshop alarm. We simply have to develop a multiple BS2-SX network, and a generic communication protocol, both of which should be flexible enough to fit a myriad of applications
.
My choice for the hardware is the relatively simple RS-485 electrical specification (the actual specification is TIA/EIA-485-A). This specification allows multiple receivers and transmitters (commonly referred to as drivers) on the same communication bus. Our system will be half-duplex (only one device transmitting at a time), and will operate using a Master-Multiple Slave protocol (a single Master unit will initiate all communication to multiple Slaves on the RS-485 bus).

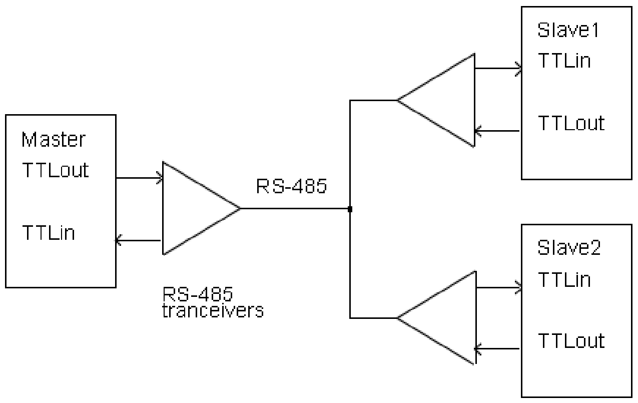
Notice that I stated previously that RS-485 was relatively simple. With short distances and low speeds, the industry standard RS-485 transceivers (both a transmitter and receiver) function well without much in the way of impedance matching or transmission line effects. For this article, the RS-485 network will be only a few inches in length so that we can concentrate on the communication protocol. Next month, we'll extend the number of BS2-SXs on the network and lengthen the cables, which may bring into play a host of gremlins.

Figure 55.1 displays a very simplified RS-485 network in block diagram format. This is pretty much what we'll have hooked up at the end of this part of the design. Again, the main reason for this is so that we can connect the parts and write a little software without tackling the larger problems that can occur when networking electronic components over long cable lengths.

The Nuts and Bolts of RS-485

RS-485 is often called a "differential-pair" multidrop network. The "differential-pair" refers to the fact that the inputs to an RS-485 receiver consist of a pair of inputs (input A and input B), and the receiver is designed to respond to voltage differences between the two inputs. Using a differential pair for detecting voltage threshold changes can significantly reduce noise effects in a system like this. The multidrop moniker refers to multiple devices residing on the network.

Figure 55.1: Simplified RS-485 block diagram



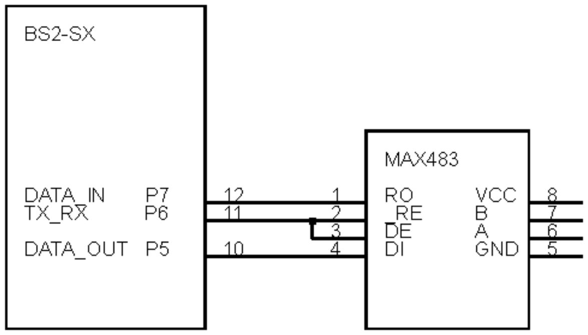
RS-485 networks have been used across distances as great as 5,000 feet. The longer the distance your RS-485 network traverses, the greater your problems become with regards to noise, ground return losses, transmission line reflections, and reduced data rates. At long distances this relatively simple network can become a significant engineering design. So try not to get too carried away with STAMP Net unless you've got some troubleshooting time on your hands.

RS-485 is an electrical specification and not a communication protocol. In other words, RS-485 is defined by how its receivers and transmitters interact electrically, and what voltage levels and current loads they can operate within. How data is sent on an RS-485 network is, to a large degree, left up to the network designer. Of course, data rates are limited by the hardware used in an RS-485 design, which may affect aspects of any communication protocol.

There are a large variety of RS-485 transceivers on the market. Some provide physical isolation for your communication system, others are lower cost, very simple transceivers. I'm partial to the use of MAXIM Integrated Products, Inc., parts. They have a solid sample policy, which can be beneficial for some of the penny-pinching hobbyists out in Stamp land. I've also had success with their parts in the past. For these reasons, I'm selecting the low-cost MAX483 RS-485 low power transceiver for this design.

The MAX483 does not provide electrical isolation for this RS-485 network. Therefore, all circuit grounds are common, and should be connected via a conductive ground wire that runs the length of the network with the communication pair (A and B inputs/outputs). On the BS2-SX side, the MAX483 is interfaced via three wires. The first two are the receiver output (RO) which sends data to the BS2-SX from the RS-485 network and the data input (DI) which receives data from the BS2-SX and places it on the RS-485 network bus. The last connection is the receiver enable (asserted with a logic low) and driver enable (asserted with logic high) pins which can be tied together and connected to a single BS2-SX output. This last connection allows the BS2-SX to select the state of the RS-485 transceiver. Figure 55.2 is representative of this connectivity.

Figure 55.2: BS2-SX interfaced to Max483



In STAMP Net, the Master unit will generally place its MAX483 in driver mode (TX_RX high), while all of the Slaves will remain in receive mode (TX_RX low), unless a command from the Master requires a response from one of the Slaves.

What's a Communication Protocol?

In today's world of high-speed telecommunication and cellular technology, I feel pretty silly describing what a communication protocol is. But I constantly hear engineers referring to electrical specifications as communication protocols. For instance, I've often heard the term RS-232 (which denotes voltage levels, as well as driver and receiver specifications in the same manner as RS-485) used to describe serial communication. When in actuality it only describes the electrical requirements of a specific communication network or bus.

A communication protocol describes the manner in which data is exchanged. The electrical specification or drivers used to connect the data generators may be part of the protocol, but they do not define it. There are certain hardware and timing issues, as well as software issues that define a communication protocol.

From a hardware standpoint, we can describe STAMP Net as an RS-485 based multi-drop network. This will be a Master-Slave system with no more than one Master and no more than seven Slaves (this number could be greatly expanded, but we'll keep it small for now). Communication will be serial in nature, and will meet the format of eight data bits, no parity, and one stop bit, with true polarity (8N1, with the start-bit defined by a logic "0" at the BS2-SX input). The data rate will be fixed at 38.4kbps. We'll be making use of the SERIN and SEROUT commands to perform the byte-by-byte transmission and reception of data.

The Master-Slave configuration is by far the easiest to implement. In this configuration, only a Master unit can initiate communication. In order for each Slave to report its current status, the Master would have to "poll" each existing unit. In a large network, this can prevent the Master-Slave concept from working. It would also be inappropriate to use a Master-Slave system if your Slave units are reporting time-critical information, such as failure conditions. But for something like a greenhouse monitor or a smaller network such as the one being designed here, the Master-Slave concept works well.

No communication protocol would be complete without a discussion of message types. Again, we're going to take the easy road, and limit the number of message types in our system to two. They will be the Master initiated Command and the Slave return of the Response. Each message will consist of eight bytes.

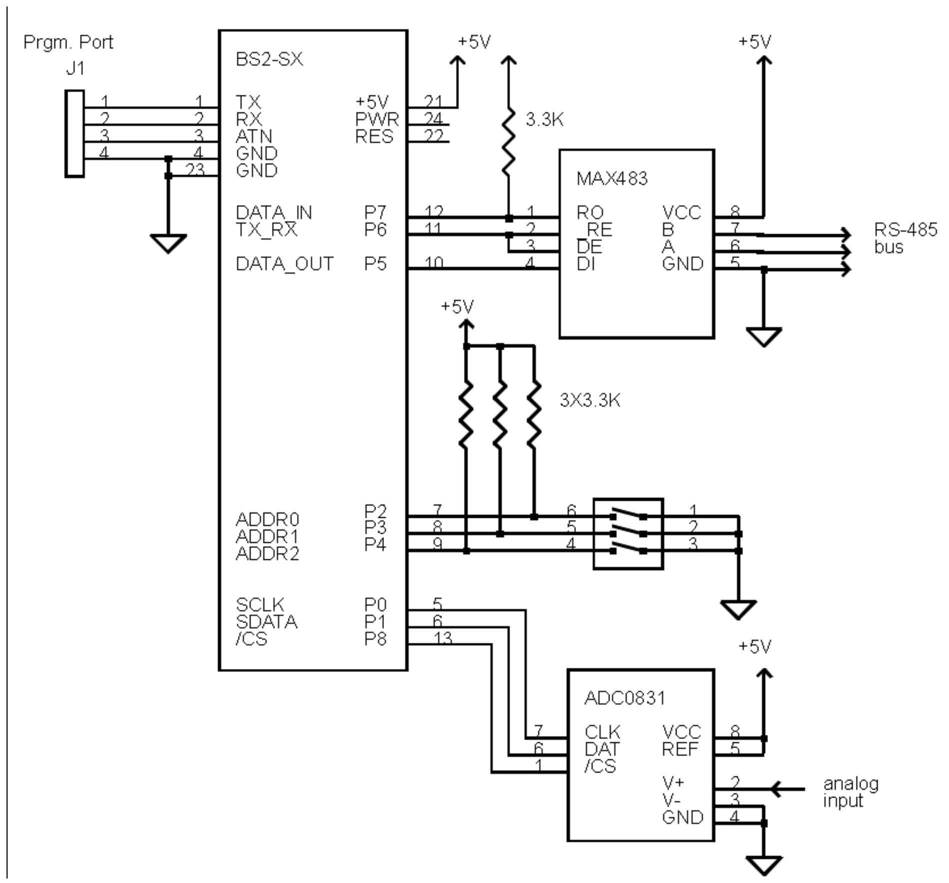
The software for the Master and Slave units will be identical in this design. In fact, the only difference between the Master and Slave units will be the address of the units. A unit with an address of "0" will default to the Master unit. Any other address (1 through 7) will default to a Slave. A Slave will always wait for communication and send its Response string to address "0." The Master, on the other hand, will poll units 1 through 7, and display their responses.

Next month, we'll add a little more functionality and a PC interface for the Master unit.

The Hardware

Since our Master and Slave units are going to be identical in software, we may as well make them identical in hardware. The addressing will be set with a three-position DIP switch. The RS-485 transceiver will be interfaced to as described by Figure 55.2, which will require an additional three I/O lines. Finally, I'm going to add an eight-bit analog-to-digital (A/D) converter, with a serial-peripheral-interface (SPI) which will require three more I/O lines. A schematic of this configuration is displayed in Figure 55.3.

Figure 55.3: BS2-SX hookup diagram with Max483 and ADC0831



The Software

The software here is broken down into two main programs. These programs are basically the starting points for developing a versatile Stamp-to-Stamp communication interface. The main program (Master_Prgm.bsx) begins by determining the address of the unit. If its address is a “0,” then the Stamp defaults to a Master and begins polling for other units on the network. Any responses are displayed via a Debug command for this program. Additionally, the communication flag register (Comm_Flag) stores bits, which indicate which STAMP Net nodes are responding. This flag register would be an easy-to-use status register indicating any units that may be having communication problems on your network.

For Slaves, Master_Prgm.bsx is used to wait for commands from the Master unit. When a command is received, it is tested for a matching address and for a correct checksum. The address requirement is necessary for any BS2-SX to ignore commands to other BS2-SXs on the network. The checksum requirement allows BS2-SX slaves to ignore any corrupted data that they may be receiving. A Slave will always respond with a "0" for both the Address and Program bytes of a response string (see Table 55.1).

Table 55.1: 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

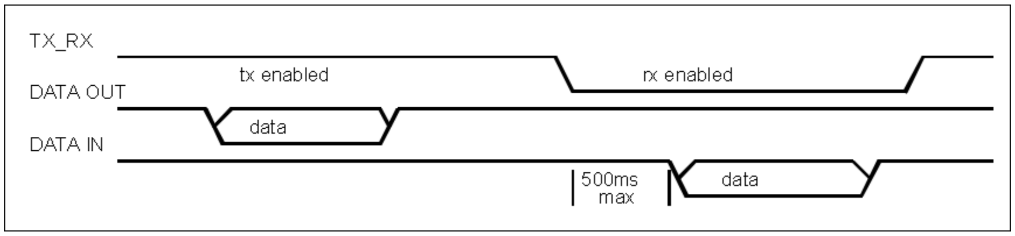
Notice that by selecting a set number of bytes for all commands and responses, I have made it easier to distinguish between commands from a Master and responses from other Slaves. And both can be processed and discarded, if need be, by the same software routine.

The second program (Analog.bsx) is the first of a set of network functional programs. This program interfaces to an ADC0831 eight-bit analog-to-digital converter; 128 samples are taken when this program is executed, and the maximum, minimum, and average measurements are returned via the RS-485 bus. This program is also responsible for responding to the Master unit.

Some communication protocol timing issues must be addressed by both pieces of software. You need to make sure that no two Stamps on the network are ever trying to transmit at the same time. For Analog.bsx, this is not a great concern. It takes long enough for the Slave BS2-SX to take all of its measurements (128 total) that the Master will be in receive mode prior to any data being sent out of the Slave executing Analog.bsx.

The time allowed for a Slave to respond to a Master unit's command is 500ms. This timing is excessive and could be reduced based on the time required for the commands issued to be processed. But this early in the game, I decided to place no major time constraints on the Master-Slave interface.

Figure 55.4: Software response timing requirement



A diagram of what you would see at the interface between a Master unit and the MAX483 is displayed in Figure 55.4.

In Closing

The STAMP Net design is by no means complete. The RS-485 interface must be tested over actual cables, and more program functions need to be added to the interface. But you can start to see the bare bones of a BS2-SX network beginning to take shape.
Next month, we'll extend out the cables to test our RS-485 network and add a little more functionality to the design. And, if time permits, we'll add a PC interface to the Master side of the software.

I can already tell that the additional program memory available in the BS2-SX will lend itself to a much more complex system than would be possible with the sturdy BASIC Stamp 2. And while it's unlikely that I'll fully utilize the capabilities of the BS2-SX in the STAMP Net design, I'm sure there's a few Stamp enthusiasts out there that are up to the challenge.

```
'Code Listing 55.1: Master_Prgm.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}
'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)

'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

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

Outs = %0000000100100000     'Set output pin values
Dirs = %0000000101100011     'Set pin direction values

Get_Address:
  Comm_Flag = %00000000
  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
'Pause 1000

If Mstr = 1 then Master_Program
Goto   Slave_Program

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

For Addr_Req = 1 to 7
       Pause          1000
       Prgm_Req       = 1
       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,500,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
       Work3 = %00000001           'Set up pointer bit
       Work3 = Work3 << Addr_Req   'Rotate "1" into Slave location
       Comm_Flag      = Comm_Flag+Work3
'Add pointer bit 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
Send_Next_Addr:
       Next
       Goto   Done_Polling
Bad_Data:
       Debug  "Checksum Invalid Addr: ",DEC Addr_Req,cr
       Goto   Send_Next_Addr
No_Data:
       Debug  "No Data Returned Addr: ",DEC Addr_Req,cr
       Goto   Send_Next_Addr
Done_Polling:
       Debug  "Comm_Flag = ", BIN8 Comm_Flag,CR
       Pause  3000
       Goto   Get_Address


'********************************************************************
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
       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
```

```
END
```

```
'Program Listing 57.2: 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)
AD_Samples     CON    128            'Number of samples taken

'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
Num_Meas       var    byte           'Storage for number of samples taken

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

Comm_Flag      = %00000000
Outs           = %0000000100100000   'Set output pin values
Dirs           = %0000000101100011   '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
```