Column #42, August 1998 by Jon Williams:

Remote Control Stamping - Part 2

Last month, we started a series of projects that we loosely referred to as "StampNet." This month, we'll continue our focus on software and communications to allow the PC and Stamp to work cooperatively. We'll also delve into a type of network control for which we don't have to build special hardware. Let's get started.

Freewheeling

During our discussion of Stamp-to-PC communications options, we decided to use event-driven communications to receive messages from the Stamp. We did this, you'll recall, because it works well in systems that specifically ask the Stamp for information, as well as those that just listen to what the Stamp has to say. Our first project fits into the latter category: it just listens to the Stamp and displays the information it receives.

It has been as hot as Hades in Texas lately, so I decided that my "freewheeling" project would monitor temperature. To do the measurement, we'll use the DS1620 from Dallas Semiconductor. Since the DS1620 has been covered by this column on more than one occasion, I'm not going to go into the details of using the chip. We'll limit our discussion to getting information to the PC and displaying it there.

Program Listing 42.1 is the BS2 code that reads the temperature from the DS1620. The DS1620 pin constants are set up for the Basic Stamp Activity Board (BSAC) [Author's note: Last month, I incorrectly referred to the BASIC Stamp Activity Board as the BSAB. Parallax calls it the BSAC and I will stick with that acronym.] You can see that the code is very straightforward. It configures the DS1620 for continuous operation, then reads the temperature and sends it (in its raw form) to the PC at one-second intervals.

Program Listing 42.2 shows the meat of the PC side of things, which is just as simple. After receiving a valid Stamp message string, the OnComm event handler passes program control to ProcessInput for data conversion.

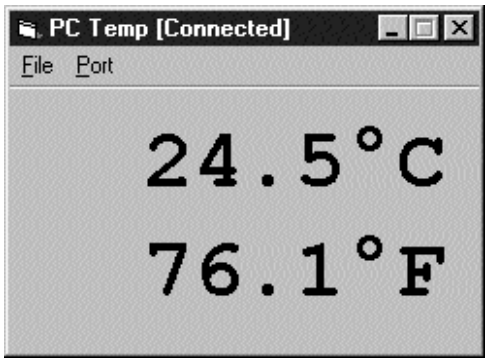
The first thing we do is grab the raw temperature data. Since the DS1620 returns a nine-bit number, we need two bytes to do the job. The first byte, hiData, will have only one of two values: $00 or $01. This is the sign bit that indicates positive or negative temperature. If hiData is $00, the temperature is positive, and all we have to do is convert the hex string to decimal and divide it by two. You'll notice that the temperature is converted to a floating point number with the CSng (convert to single) function. The division by two is necessary since the DS1620 reports temperature in half-degree increments.

If hiData is $01, the temperature is negative and has been returned in 2's complement form. We convert from 2's complement by XORing with 255 and then adding 1. This time, we divide by -2 since our sign bit told us this was a negative temperature.

Here's the best part: conversion and display. The PC makes this task trivial. A couple of labels on the form and the VB Format function make for a nice display. You can see that the Celcius-to-Fahrenheit conversion is built right into the display code. Figure 42.1 shows the program in action. Obvious extensions of this program include multicolor graphing, long-term data storage, and analysis.

The basic theme of this project is to let the Stamp gather raw data from your sensors and let the PC convert the data to the appropriate engineering units and provide a nice display. The Parallax application notes for the Stamp 1 show how to interface the Stamp to a thermistor — a good candidate for this kind of program due to its non-linear conversion requirements.

Figure 42.1: Visual BASIC display of temperature



Stamp-Based Home Control

With the Stamp 2, Parallax introduced several new features to the PBASIC programming language. Among the BS2 features is XOUT, a command the sends control codes to X-10 appliance and lamp modules.

In application, the BS2 is connected to your home electrical system with a special interface called a PL-513 or the TW-523. The latter supports the reception of X-10 codes, but since the Stamp does not support this feature, there is no point in spending the extra money for the TW-523. Parallax carries the PL-513. You can get lamp and appliance modules at Radio Shack or your local home center.

X-10 modules use a two-digit addressing scheme: house code and unit code. On the X-10 units, house code is a letter between "A" and "P" (inclusive). The Stamp represents this range with a nibble value of zero to 15. The idea of the house code is that you would use one letter while your neighbor would use another. This would prevent you from creating problems for each other.

The unit code on the X-10 module is from one to 16. As with the house code, internally the Stamp represents the unit code with a value from zero to 15. We'll take advantage of this in our PC program by packing nibbles into a byte for transmission.

The BS2 supports six X-10 commands: Unit On, Unit Off, All Lamps On, All Units Off, Dim, and Bright. Not all commands work with all X-10 modules. All Lamps On, for example, will turn on all lamp modules with the same house code, but appliance modules are not affected. Likewise, with Dim and Bright: they only work with lamp modules. All

Units Off, however, will shut down anything with the same house code.

Program Listing 42.3 is a BS2 program that accepts X-10 commands from the PC and sends the appropriate codes with XOUT. As with last month's project, I downloaded this to a BS2 that is plugged into the BSAC. (Of the many conveniences of the BSAC, it has a built-in RJ-11 jack for connecting the Stamp to the PL-513 or TW-523 with a telephone cable.) Structurally, this program is identical to last month's program with a couple of updates and the ability to handle the X-10 stuff.

The first thing you'll notice is that the variable pcData has been promoted from a byte to a word. We need the extra byte to send all the X-10 data and will use this next month when sending 12-bit data to a DAC. Interestingly, however, we split it up for receipt from the PC. The reason for this has to do with the Visual Basic integer type. It is a signed type, that is, bit 15 is a sign bit so we can't manipulate the VB integer in the same manner as a PBASIC word. It's not a big problem, though. With the ability to map variables, we can easily construct a word from the two bytes via our variables declaration section.

You'll also notice that I've mapped several variables having to do specifically with the X-10 stuff. For those of you that are new to PBASIC, this mapping does not use any more variable space in the Stamp 2. It simply gives my generic variables names that make the actual program code easier to read — almost self-commenting.

Working our way into the code, we see that the program waits for the data from the PC (prefaced with $55), then checks the address and command to make sure that the data string is intended for this node (the master node) and that the command is X-10 ($F).

This may not all make sense now, particularly the addressing part. Next month, we will pass on information not intended for the master node to a remote Stamp. Since we've planned for this eventuality, designing the code to handle it now will make folding X-10 control into our fully-blown StampNet network that much easier.

The PC sends a code of one to five to indicate which X-10 function to execute. We use the BRANCH command to route the program to the proper subroutine. The first four are very straightforward; the dim command, however, requires a little bit of discussion.

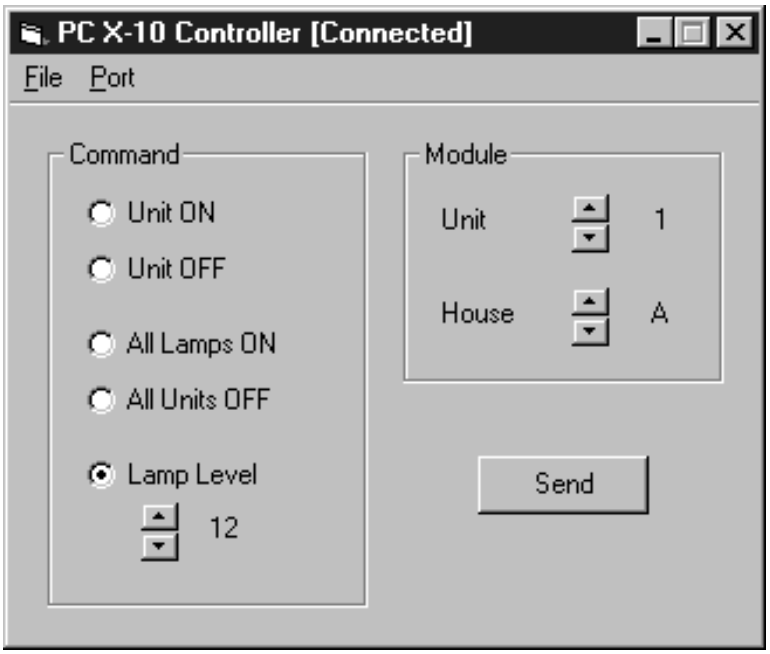
Dim and Bright are relative to the current brightness of the lamp. Since it's not always possible to know the current state of a lamp dimmer, the Parallax manual suggests that the easiest implementation is to turn off the lamp, then dim it down to the desired level. That's right: turn off, then dim down. This seems a bit peculiar at first, and yet it works. I spent about a half hour experimenting and this is the easiest method. Valid dimming

values are from one (brightest) to 19 (just barely on).

The last part of the code sends an acknowledgment to let the PC know that the command has been carried out. The PC expects this string and will alert the user if it doesn't show up within a reasonable amount of time.

Take a look at Figure 42.2. This is the PC side of our X-10 control project. The form is divided into two frames: Command and Module. Radio buttons are used to select the X-10 function and spin controls are used to set the lamp level, and house and unit codes. The nice thing about using the PC as our interface is the ability to make the controls organized and intuitive.

Figure 42.2: X10 controls



The lamp level, for example, has a range from 1 (dim) to 19 (bright) — what the user would expect (big number = bright light). Since we're using the dim function of the XOUT command, this value gets inverted before sending to the Stamp.

And note that the unit and house codes are physically laid out just as they are on an X-10 module. The associated spin controls have Stamp-compatible values, while their labels are converted to something the user expects (i.e., letters for house codes). These are

simple things to do in code and will make your PC programs easier to use.

Program Listing 42.4 is the Visual Basic code that corresponds to clicking the "Send" button. You can see that the house code (updnHouse.Value) is embedded in the message part of our data string. The house code is added to $F0 (&HF0 in VB), the command for X-10 functions. The unit code (updnUnit.Value) is placed in the high nibble of the first data byte and the X-10 function (1-5) is embedded in the low nibble of this byte. Finally, the corrected lamp level is placed in the second data byte.

The next two lines of code look harmless, yet serve a very important function. This project uses a timer to let you know if the Stamp isn't there. Here's how it works. When the command is sent to the Stamp, the timer interval value (in milliseconds) is loaded and the timer is enabled. Once enabled, the timer will count down until it's disabled or the interval value reaches zero. If the proper acknowledgment is received, the timer is disabled and no alarm is generated. Otherwise, a dialog box is displayed to let the user know that the Stamp is not talking back. When you look through the code, you'll see that it takes quite a bit longer to send dim commands than basic on/off, so the time-out value is larger.

Where do you go from here? Well, that's up to you. Sure, there's a lot of off-the-shelf X-10 control software available, but the ability to use the Stamp 2 as an interface gives you some serious advantages. You could, for example, use the Stamp to monitor environmental conditions (light, temperature, etc.) and use X-10 modules to adjust lamps and appliances accordingly. Give this program a try; it really is a lot of fun. Kind of like a blown-up version of a blinking LED program — it'll make you smile when you can control lights in another room from a PC program that you wrote yourself.

```
' Program Listing 42.1
' Nuts & Volts: Stamp Applications, August 1998

' -----[ Title ]-----------------------------------------------------
'
' File...... PC1620.BS2
' Purpose... Dallas DS1620 <-> Stamp II -> PC (Visual Basic)
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' WWW....... http://members.aol.com/jonwms
' Started... 10 JUN 1998
' Updated... 10 JUN 1998


' -----[ Program Description ]---------------------------------------
'
' This program reads the temperature from the Dallas DS1620 and sends it
' to the PC for conversion and display.


' -----[ Revision History ]------------------------------------------
'
' 10 JUN 98 : Rev 1 complete


' -----[ Constants ]-------------------------------------------------
'
True    CON     1
False   CON     0

' DS1620 pins
' -- setup for BSAC
'
Rst     CON     13                      ' DS1620.3
Clk     CON     14                      ' DS1620.2
DQ      CON     15                      ' DS1620.1

' DS1620 commands
'
RdTmp   CON     $AA                     ' read temperature
WrTHi   CON     $01                     ' write TH (high temp register)
WrTLo   CON     $02                     ' write TL (low temp register)
RdTHi   CON     $A1                     ' read TH
RdTLo   CON     $A2                     ' read TL
StartC  CON     $EE                     ' start conversion
StopC   CON     $22                     ' stop conversion
WrCfg   CON     $0C                     ' write configuration register
RdCfg   CON     $AC                     ' read configuration register
CPU     CON     %10
NoCPU   CON     %00
```

```
OneShot CON     %01
Cont    CON     %00

Baud96  CON     84                      ' serial baud rate (to PC)
SIOPin  CON     16                      ' use programming port


' -----[ Variables ]-------------------------------------------
'
tmpIn   VAR     Word                    ' 9-bit temp input from DS1620


' -----[ EEPROM Data ]-----------------------------------------
'


' -----[ Initialization ]--------------------------------------
'
Init:   LOW Rst
        HIGH Clk
        PAUSE 100

' Initialize the DS1620
' - use with CPU in free run mode
'
I_1620: HIGH Rst
        SHIFTOUT DQ,Clk,LSBFIRST,[WrCfg,CPU+Cont]
        LOW Rst
        PAUSE 10
        HIGH Rst
        SHIFTOUT DQ,Clk,LSBFIRST,[StartC]
        LOW Rst


' -----[ Main Code ]-------------------------------------------
'
Main:   ' get temp from DS1620
        HIGH Rst
        SHIFTOUT DQ,Clk,LSBFIRST,[RdTmp]
        SHIFTIN DQ,Clk,LSBPRE,[tmpIn\9]
        LOW Rst

        ' send to PC
        SEROUT SIOPin, Baud96, [hex2 $55, hex2 $01, hex4 tmpIn, 13]

        PAUSE 1000
        GOTO Main


' -----[ Subroutines ]-----------------------------------------
'
```

```
' Program Listing 42.2
Public Sub ProcessInput()
  Dim hiData As String
  Dim loData As String
  Dim tempC As Single
  Dim rawNeg As Integer

  hiData = Mid$(buffer, 5, 2)
  loData = Mid$(buffer, 7)

  If hiData = "00" Then ' positive temp
    tempC = CSng(HexToDec(loData)) / 2
  Else ' negative temp
    ' two's compliment
    rawNeg = (HexToDec(loData) Xor 255) + 1
    tempC = CSng(rawNeg) / -2
  End If

  ' display
  lblTempC.Caption = Format(tempC, "##0.0°C")
  lblTempF.Caption = Format(tempC * 9# / 5# + 32#, "##0.0°F")
End Sub
```

```
' Listing 42.3
' Nuts & Volts: Stamp Applications, August 1998
' ----[ Title ]---------------------------------------------------------
'
' File...... PCX10.BS2
' Purpose... PC-based X-10 Control via the Stamp II
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' WWW....... http://members.aol.com/jonwms
' Started... 28 JUN 1998
' Updated... 28 JUN 1998


' ----[ Program Description ]-------------------------------------------
'
' This program X-10 commands from the pc using the StampNet communications
' protocol. Actual X-10 control is via the XOUT command.
'
' Message structure:
'
'      $55           header
'      addr          node address (must be 1 for X-10)
'      msg           HighNib = "F" for X-10, HowNib = house code
'      pcHigh        HighNib = unit code, LowNib = pc X-10 command
'      pcLow         lamp level for dimmer modules
```

```
' ----[ Revision History ]------------------------------------------------
'
' 28 JUN 98 : Rev 1


' ----[ Constants ]---------------------------------------------------------
'
Baud96  CON    84                   ' serial baud rate (to PC)
SIOPin  CON    16                   ' use programming port
                                    '  - couple ATN with capacitor
mPin    CON    0                    ' X-10 comm pin
zPin    CON    1                    ' X-10 zero-cross pin


' ----[ Variables ]---------------------------------------------------------
'
addr    VAR    Byte                 ' node address
msg     VAR    Byte                 ' message
cmd     VAR    msg.Nib1
channel VAR    msg.Nib0
pcData  VAR    Word                 ' data from pc
pcHigh  VAR    pcData.HighByte
pcLow   VAR    pcData.LowByte
sData   VAR    Word                 ' return data
sDHigh  VAR    sData.HighByte
sDLow   VAR    sData.LowByte

' X-10 variable mapping
hCode   VAR    msg.Nib0             ' X-10 house code (0-15)
uCode   VAR    pcHigh.HighNib       ' X-10 unit code (0-15)
x10     VAR    pcHigh.LowNib        ' X-10 command (1 - 5)
lmpLvl  VAR    pcLow                ' lamp level (1-19)


' ----[ EEPROM Data ]-------------------------------------------------------
'


' ----[ Initialization ]----------------------------------------------------
'


' ----[ Main Code ]---------------------------------------------------------
'
Main:   ' wait for message from PC
        SERIN SIOPin, Baud96, [WAIT ($55), addr, msg, pcHigh, pcLow]

        ' ignore bad address or non X-10
        IF (addr <> 1) | (cmd <> $F) THEN Main
```

```
        BRANCH x10,[Main, Unit1, Unit0, All1, All0, SetLvl]
        ' ignore if invalid x-10 command
        ' zero is not valid
        GOTO Main


' ----[ Subroutines ]-----------------------------------------------------
'

' turn selected unit on
Unit1:  XOUT mPin, zPin, [hCode\uCode\2,hCode\unitOn]
        GOTO Rspnd

' turn selected unit off
Unit0:  XOUT mPin, zPin, [hCode\uCode\2,hCode\unitOff]
        GOTO Rspnd

' turn all lamps on
All1:   XOUT mPin, zPin, [hCode\uCode\2,hCode\lightsOn]
        GOTO Rspnd

' turn all units off
All0:   XOUT mPin, zPin, [hCode\uCode\2,hCode\unitsOff]
        GOTO Rspnd

' set light level
SetLvl: XOUT mPin, zPin, [hCode\uCode]
        XOUT mPin, zPin, [hCode\unitOff\2,hCode\dim\lmpLvl]
        GOTO Rspnd

Rspnd:  ' output to PC (x-10 signature)
        SEROUT SIOPin, Baud96, ["5501F000", 13]
        GOTO Main
```

```
' Program Listing 42.2
Private Sub btnSend_Click()
  SendMsg 1, &HF0 + updnHouse.Value, updnUnit.Value * 16 + x10, lamp
  ' start the response timer
  Timer1.Interval = timeOut
  Timer1.Enabled = True
End Sub
```