



Column #123 July 2005 by Jon Williams:

Getting Hot, Hot, Hot

I think it's fair to say that my friends would tell you I'm a bit of a quirky guy. I accept that; I am what I am. One of my many quirks – one that makes me laugh at myself – is how freakishly sensitive I am to temperature. I probably adjust the thermostat in my home 15 to 20 times a day – and that includes the night too (if I have to get up for drink of water, I'm visiting the thermostat). Well, now that it's summer in north Texas, it's getting hot (as it is in most of the northern hemisphere) and it's probably time for more experiments with temperature.

Like the BASIC Stamp, the Maxim/Dallas DS1620 has been around a long time and has been a big part of my temperature-based projects. Yet in all this time, I had never explored the high-resolution use of the DS1620. “High resolution?” you ask. Yeah. With just a little bit of extra work we can get temperature resolution to 0.05 degrees Celsius (0.09 degrees Fahrenheit) from our old stand-by. How is this possible?

You see, the DS1620 actually measures temperature through the use of a couple temperature-controlled oscillators that drive a counter. When one oscillator rolls over within the period determined by the other oscillator, the temperature count is incremented (from the base of -55C). The key for us is that the fractional portion of the temperature can be determined by

examining the count left over at the end of the conversion period and comparing it to the number of counts per degree (called the slope – this value is used to linearize the natural non-linear behavior of the oscillators).

Before we get to the high-resolution calculation, let's go back to what we know and use the standard calculations first. What we will do differently is configure the DS1620 so that it converts temperature only when requested (we've typically set it up for continuous conversion), and we'll recode for PBASIC 2.5 – which you'll see makes things dramatically easier than before. Figure 123.1 shows the connections to the DS1620. For those of you that are new, don't leave the 1K resistor out of the circuit. The DQ pin is bi-directional and resistor protects the BASIC Stamp and the DS1620 in the event that both IO pins are made outputs and driven in opposite directions (one high, one low – which would cause a short circuit without the resistor).

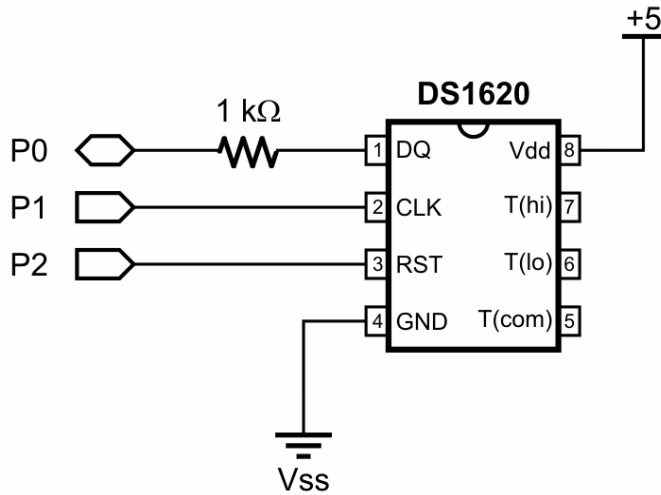


Figure 123.1: DS1620 Interface with BASIC Stamp

Let's get to the initialization. As you can see, it's simpler than what we've used in the past as we're just configuring for use with a CPU and in one-shot mode. We start by activating the DS1620 (Reset pin is made high), then writing %11 to the configuration register. When that's done we can deactivate the DS1620 by taking the DS1620 low.

Setup:

```
HIGH DsRst
SHIFTOUT DsDQ, DsClk, LSBFIRST, [WrCfg, %11]
LOW DsRst
PAUSE 10
```

Just a note on the DS1620 Reset pin: It does more than select the device we're addressing; it also terminates a communication "burst" with the host. I bring this up so that you don't think you can tie that line high when you're just using one DS1620 in a project.

Okay, you may notice that the program doesn't run correctly the first time – we have to cycle power for the new configuration to "take." Now that we've configured the DS1620 for one-shot mode, let's start a temperature conversion.

Get_Temp:

```
HIGH DsRst
SHIFTOUT DsDQ, DsClk, LSBFIRST, [StartC]
LOW DsRst
```

How do we know when the conversion cycle is done? Well, we could take the easy way out and just pause for about a second, but the DS1620 will actually tell us when it's finished.

```
DO
  HIGH DsRst
  SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdCfg]
  SHIFTIN DsDQ, DsClk, LSBPRE, [tempIn\8]
  LOW DsRst
LOOP UNTIL (tempIn.BIT7 = 1)
```

The end of the temperature conversion cycle is signaled by bit 7 of the configuration register. What this loop of code does is read that register until bit 7 goes high. Here's a great example of how PBASIC 2.5 features make BASIC Stamp programming so much more elegant.

When the conversion is complete we can read the temperature just as we've done in the past:

```
HIGH DsRst
SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdTmp]
SHIFTIN DsDQ, DsClk, LSBPRE, [tempIn\9]
LOW DsRst
```

Column #123: Getting Hot, Hot, Hot

And, finally, we can calculate the temperature to 0.5 degrees Celsius resolution:

```
IF (sign = 0) THEN
  tC = tempIn * 5
  tF = tC * 9 / 5 + 320
ELSE
  tC = tempIn | $FF00 * 5
  tF = 320 - ((ABS tC) * 9 / 5)
ENDIF
```

Remember that the DS1620 returns a 9-bit temperature value, and that the LSB (bit 0) is equal to 0.5 degrees Celsius. The first thing to check is the sign bit (bit 8) – when this bit is one, the temperature is negative. In most of our projects it won't be, so let's start at the top. We multiply the value returned by the DS1620 by five to convert the temperature into tenths. So if the temperature is 23.5 degrees C, we'll end up with 235 in the variable tC. Now we can convert to Fahrenheit using the standard formula $F = C \times 1.8 + 32$. As we're working in tenths, we have to multiply 32 by 10 as well to keep things intact.

Now, let's look at handling negative temperatures (when bit 8 of tempIn is 1). First, do this in your BASIC Stamp Editor:

```
DEBUG IBIN16 -55
```

Some will be surprised by the result: %111111111001001. The reason for this is that the BASIC Stamp stores negative numbers in two's-complement format. So does the DS1620. The thing is, the DS1620 only returns nine bits, so we have to "fix" that by setting the upper bits of tempIn before moving on with the rest of the calculations. We do this by ORing tempIn with \$FF00.

Another thing to note is that we cannot use division with negative values, hence the use of ABS (absolute value) in the Fahrenheit calculation. Using the ABS function makes the tC value positive in the calculation, so adjust by subtracting the tC portion from 320 – cha-ching, everything is correct now.

Finally, let's put the temperature up in the Debug Terminal window:

```
Show_C:
  DEBUG CRSRXY, 6, 2,
    (tC.BIT15 * 13 + 32),
    DEC (ABS tC / 10), ".", DEC1 (ABS tC),
    DegSym, CLREOL

Show_F:
  DEBUG CRSRXY, 6, 3,
    (tF.BIT15 * 13 + 32),
    DEC (ABS tF / 10), ".", DEC1 (ABS tF),
    DegSym, CLREOL
```

This code starts by examining bit 15 of the value – when bit 15 is one the value is negative. From this bit we create a negative sign or space (when positive) to precede the value. The rest is simple; we'll divide the [absolute] value by 10 for the whole portion, print a decimal point, and then use DEC1 for the final digit to display the tenths.

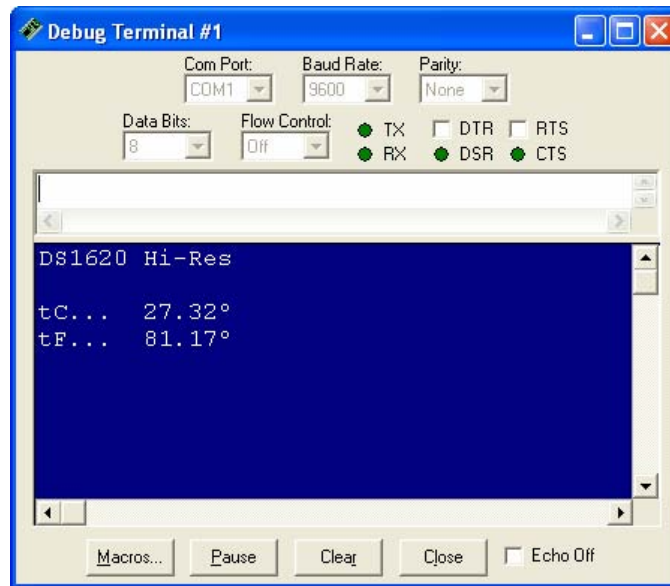


Figure 123.2: Displaying the DS1620 Temperature in Debug Terminal

Going Higher

To get high-resolution temperature from the DS1620 we will proceed as before and then read two additional values: the count remaining and the slope. To do this we will add the following code after reading the temperature value:

```
HIGH DsRst
SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdCntr]
SHIFTIN DsDQ, DsClk, LSBPRE, [cRem\9]
LOW DsRst

HIGH DsRst
SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdSlope]
SHIFTIN DsDQ, DsClk, LSBPRE, [slope\9]
LOW DsRst
```

The first section reads the count register, the second reads the slope accumulator. With these values we can calculate high resolution temperature with this equation:

$$tC = 0.25 + (\text{slope} - \text{counts}) / \text{slope})$$

Note that tC in the equation above is the whole value from the DS1620 – the half-bit is dropped (as this was determined by estimation inside the DS1620). Here's how we implement the high-resolution calculation resolution in PBASIC:

```
IF (sign = 0) THEN
  tC = (tempIn / 2) * 100
  tC = tC - 25 + (slope - cRem * 100 / slope)
  tF = tC * 9 / 5 + 3200
ELSE
  tC = (tempIn / 2) | $FF00 * 100
  tC = tC - 25 + (slope - cRem * 100 / slope)
  tF = 3200 - ((ABS tC) * 9 / 5)
ENDIF
```

In order to deal with the 0.25 value in the equation, as well as take advantage of the increased resolution offered, everything is converted to hundredths. Other than that, you can see that the calculation is quite straightforward and with an adjustment to our display code (for hundredths) the output we get looks like that in Figure123.2.

A Little Help from a Friend

That was actually pretty easy, wasn't it? What about those times when we have a sensor that requires complex calculations to convert its raw output to something we can use? After I was satisfied with the hi-res version of the DS1620 program I took note of the Micromega Corporation uM-FPU (V2.0) chip sitting on my desk. This device, kindly sent to me by Cam Thompson, is a floating-point math coprocessor that is designed to assist small micros like the BASIC Stamp. I've had the thing for several months; I thought it time to give it a whirl.

Following my own frequent advice, I cracked open the uM-FPU docs and read through them. Holy smokes, Batman, this little dude is a hand-full. After my first read I thought my eyes were bleeding and my brain had exploded! Okay, all kidding aside, it's not terribly complicated, but it is very sophisticated and if one doesn't proceed deliberately things can get out of hand in a big hurry.

Think about it, the uM-FPU is a coprocessor for floating-point mathematics – something that we all [should] know is NOT a trivial process. Floating-point requires a gigantic amount of processor resources; hence most micros don't have FP built in. In fact, it wasn't all that long ago when our PC processors started coming with FP built-in. Many of us remember the good-old-days when we had to crack open our PC to add a floating-point coprocessor (my first was an 80287 for my IBM PC Model 50) to speed up math-intensive applications like CAD.

Using a coprocessor for a small micro like the BASIC Stamp makes sense to me – most of my projects do not require FP math, so why waste the resources when I can add FP only when needed? But that's just my opinion, and I know that many of you think differently. For those that are looking for a way to add FP math to your BASIC Stamp projects, we're going to work through converting our DS1620 project for use with the uM-FPU. Now, I will admit without reservation, that this project is not even coming close to scratching the surface of capabilities of the uM-FPU; but it will get you going, and will prepare you for more complicated tasks.

Figure 123.3 shows the connections for the uM-FPU using SPI mode. The uM-FPU is fairly flexible in its connections and has separate input and output data pins (SIN and SOUT) for micros that can't use the same IO pin for input an output. Since the BASIC Stamp can do that without any problems, we simply put a 1K resistor between those pins to prevent any conflicts.

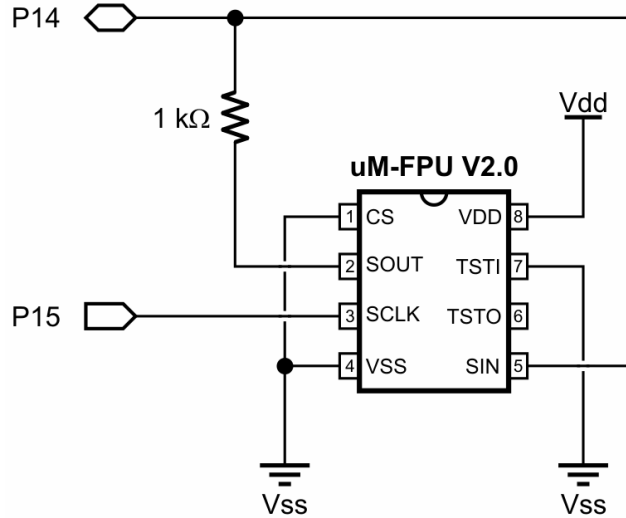


Figure 123.3: uM-FPU 2.0 to BASIC Stamp Connection

Notice that the uM-FPU has a CS pin. This is not a Chip Select as we might first assume. What this pin actually does is configure the communication mode of the uM-FPU. When tied low (as we're doing), the uM-FPU uses SPI communications; when tied high, it uses I2C communications (yes, clock and SIN/SDA must be pulled up). The latter mode is convenient when using the BS2p family and when there's already an I2C bus in the project.

Getting to the meat of things, the uM-FPU is a processor with its own language. Briefly, the device uses 16 32-bit registers to hold values, and two pointers (A and B) to direct the operations. If, for example, we wanted to multiply register 1 by register 2, then add register 3 and place the result in register 4, the uM-FPU instructions would look like this:

```

SELECTA+4
XOP, LEFT
FSET+1
FMUL+2
XOP, RIGHT
FSET
FADD+3
    
```

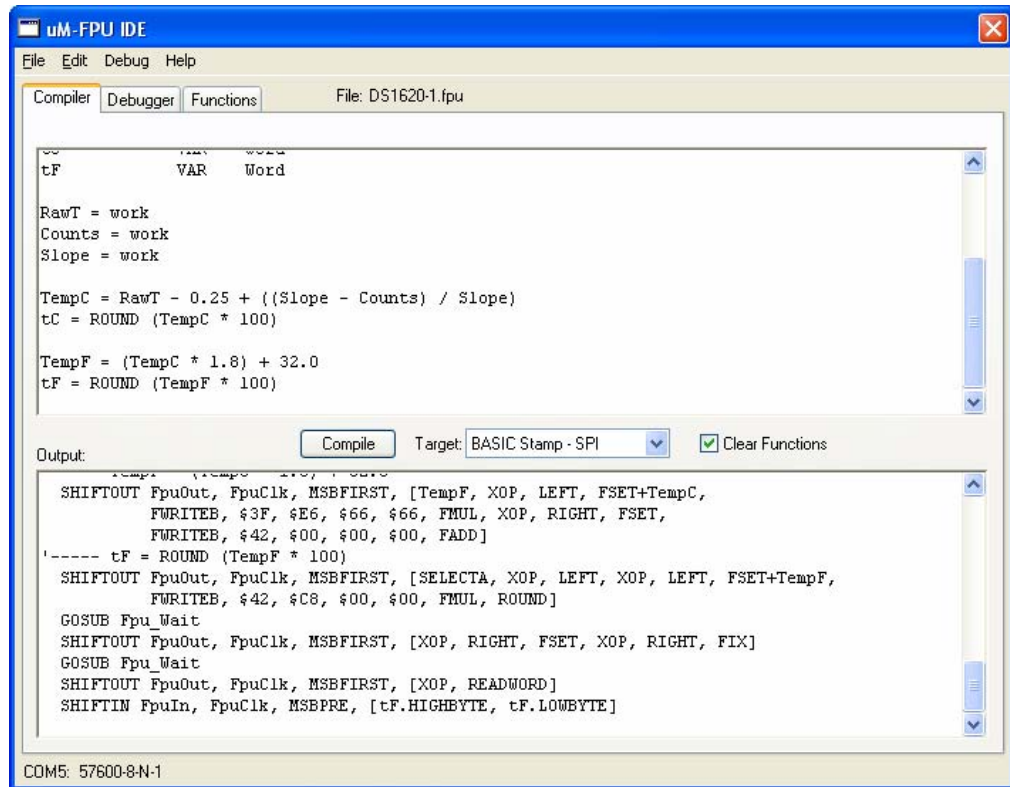


Figure 123.4: uM-FPU IDE

I don't know about you, but my plate is pretty full and I really don't have time to learn an arcane language for a chip that I wouldn't use very frequently anyway. I nearly scrapped the idea of using the uM-FPU until I remembered a comment in the docs about an IDE for the uM-FPU. And since Cam was kind enough to send me the chip, I thought I should at least give that a look before walking away.

Hallelujah! What a difference a simple program can make in my attitude toward the uM-FPU. The IDE makes it so we don't have to learn the uM-FPU language – it will take very traditional looking code and covert it to uM-FPU instructions for us. The program even lets us select the compiled output format, including BASIC Stamp running in SPI mode. Now were talking!

Column #123: Getting Hot, Hot, Hot

So, before you get to involved in the uM-FPU programming commands, download the IDE and give it a try – it will save you hours of frustration. Figure 123.4 shows the IDE with the first pass DS1620 code loaded up. You can see our input in the top window and the compiled output (set for BASIC Stamp SPI) in the bottom window. Let's look at the input code.

```
RawT      EQU      F1
Counts    EQU      F2
Slope     EQU      F3
TempC     EQU      F4
TempF     EQU      F5

work      VAR      Word
tC        VAR      Word
tF        VAR      Word

RawT = work
Counts = work
Slope = work

TempC = RawT - 0.25 + ((Slope - Counts) / Slope)
tC = ROUND (TempC * 100)

TempF = (TempC * 1.8) + 32.0
tF = ROUND (TempF * 100)
```

As with any other program, we start by defining storage space – in this case were going to name the floating point registers required as well as our own PBASIC variables. By using our PBASIC variables in the uM-FPU IDE code, the output will be ready to paste right in to our BASIC Stamp application.

After the definitions we have to transfer data from the BASIC Stamp to the uM-FPU. Since we're now using internal registers, we can use a single variable (called work) in our BASIC Stamp code. We'll see how all this meshes in just a bit. With the raw values in place, the calculations match what we find in the DS1620 docs. Now remember that the BASIC Stamp uses integers, so what we'll do is convert the temperature to hundredths and then to fixed point (with ROUND) before going back to the BASIC Stamp.

After clicking on the Compile button we'll code that is ready to paste into our BASIC Stamp program. What I should also point out is that the uM-FPU comes with a template program that includes several useful subroutines. What I actually did was add my DS1620 interface to the uM-FPU template to get everything together. I've included both the template and the three versions of the DS1620 program in the download ZIP file.

Our task now is to paste the output code from the IDE into our application where it's needed. For example, this line of uM-FPU code:

```
RawT = work
```

... compiles to:

```
' ----- RawT = work
SHIFTOUT FpuOut, FpuClk, MSBFIRST,
        [RawT, LOADWORD, work.HIGHBYTE,
        work.LOWBYTE, FSET]
```

Remember that we need to truncate the half-degree bit and fix the sign (if required) before sending the value to the uM-FPU. After repeating this process for other raw values we get to the calculations. To keep things simple, let's just look at the high-resolution Celsius calculation:

```
' ----- TempC = RawT - 0.25 + ((Slope - Counts) / Slope)
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [TempC, FSET+RawT,
        FWRITEB, $BE, $80, $00, $00, FADD, XOP, LEFT,
        XOP, LEFT, FSET+Slope, FSUB+Counts, XOP, RIGHT,
        FSET, FDIV+Slope, XOP, RIGHT, FADD]
```

Wow, that's a mouthful, isn't it? You can see why the uM-FPU IDE is such a Godsend – I'd hate to have to figure this out on my own. But wait, we're not done – we've still got to convert the temperature to hundredths, back to fixed point, and pull it back into the BASIC Stamp.

```
' ----- tC = ROUND (TempC * 100)
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [SELECTA]
GOSUB Fpu_Wait
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, LEFT, XOP, LEFT,
        FSET+TempC, FWRITEB, $42, $C8, $00, $00, FMUL, ROUND,
        XOP, RIGHT, FSET, XOP, RIGHT, FIX]
GOSUB Fpu_Wait
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, READWORD]
SHIFTIN FpuIn, FpuClk, MSBPRES, [tC.HIGHBYTE, tC.LOWBYTE]
```

Column #123: Getting Hot, Hot, Hot

At this point the variable tC holds the temperature (in Celsius) and we can display it as we did in the original version of the hi-res program. Some of you will [logically] wonder why we would go through so much trouble for calculations that weren't that tough to start with. Well, of course we wouldn't with the DS1620. Remember that our purpose here was to take something we know (DS1620) and use it to help us learn something new (uM-FPU).

I'm sure by now you've also noticed that it does take a fair bit of PBASIC code to execute calculations inside the uM-FPU. And what happens when we want to do something really complicated? Thankfully, Cam thought of that and has provided a solution. You see, another thing the uM-FPU IDE can do for us is download the calculations to the chip. After that, all we have to do is send the raw data, request a specific calculation be executed, then retrieve the desired data. To do this we have to prep the chip by removing it from our application and connecting it to the PC as shown in Figure 123.5. Since I was using the PDB for my experiments I used the spare serial port. And as a test, I also connected using the USB2SER adapter and found that it works just fine too.

The uM-FPU code for embedded calculations changes a bit. Let's have a look.

```
RawT          EQU      F1
Counts        EQU      F2
Slope         EQU      F3
TempC         EQU      F4
TempF         EQU      F5
TempCL        EQU      L6
TempFL        EQU      L7
work          VAR      Word

#FUNCTION 0 Calc_TC
  TempC = RawT - 0.25 + ((Slope - Counts) / Slope)
  TempCL = ROUND(TempC * 100)
#END

#FUNCTION 1 Calc_TF
  TempC = RawT - 0.25 + ((Slope - Counts) / Slope)
  TempF = (TempC * 1.8) + 32.0
  TempFL = ROUND(TempF * 100)
#END

RawT = work
@Calc_TC
work = TempCL
@Calc_TF
work = TempFL
```

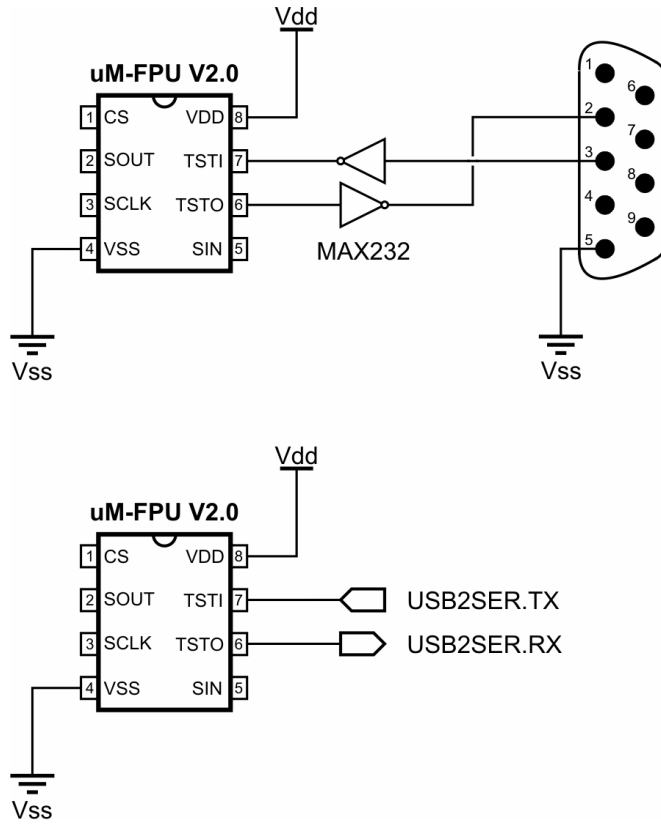


Figure 123.5: uM-FPU to PC Serial Port Connection

Note that we've added a couple fixed point registers (L is for Long, the 32-bit fixed format) to hold the final result of our calculations. And in order to embed and access the temperature calculations, we must define them as functions as shown in the listing. In our case, Function 0 is called Calc_TC and the Function 1 is called Calc_TF.

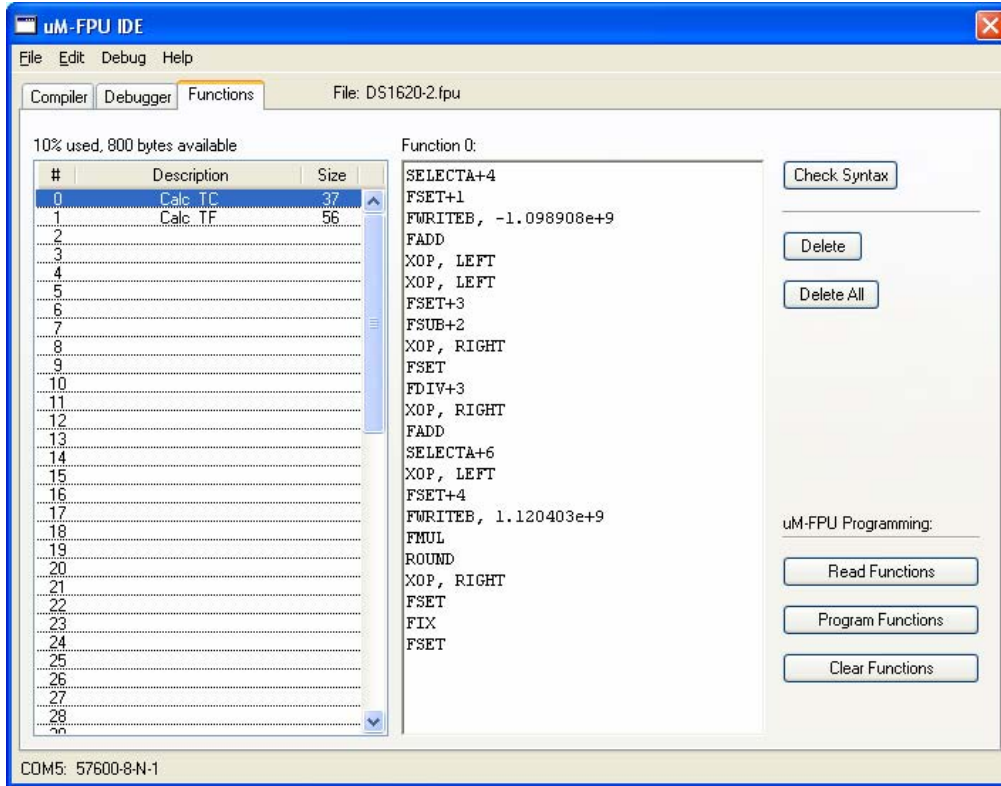


Figure 123.6: uM-FPU IDE Functions

When you look at the IDE Output box you'll see that there is no code for the functions; the only thing we have moving data between the BASIC Stamp and the uM-FPU. Click on the IDE "Functions" tab and should get something like shown in Figure 123.6. With the uM-FPU connected to our computer we can click on the "Program Functions" button to move the code into the uM-FPU.

To be honest, I think this is the strongest suit of the uM-FPU. We can preprogram a wide variety of calculations and then call them as required. I like that. This example is simple, but Cam has done some really neat things with the uM-FPU, including the calculations for inverse kinematics for a robot arm!

After the uM-FPU is programmed with the calculations, put it back into the DS1620 project and replace the code that actually performed the high-resolution Celsius calculation with this:

```
' ----- @Calc_TC
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, FUNCTION]

' ----- work = TempCL
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [TempCL]
GOSUB Fpu_Wait
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, READWORD]
SHIFTIN FpuIn, FpuClk, MSBPRES, [tC.HIGHBYTE, tC.LOWBYTE]
```

See how much cleaner our program becomes when we store the calculations in the uM-FPU? Yeah, that's the way I like it.

I think that's about enough for this month, don't you? With summer in full swing (and winter for our friends down under), we now have the tools to tell the temperature with much more precision than we've used in the past. And we also have a cool new tool to help us with complex calculations. Be sure to visit the Micromega web site for IDE updates and lots of neat applications notes for the uM-FPU. Now that you've started, the rest should get pretty easy.

Until next time, stay safe, have fun, and Happy Stamping!

Additional resources:

Micromega Corporation
www.micromegacorp.com

Column #123: Getting Hot, Hot, Hot

```
' =====
'
' File..... uM-FPU Template.BS2
' Purpose.... Micromega FPU V2.0 Interface
' Author..... Jon Williams -- Parallax, Inc.
' E-mail..... jwilliams@parallax.com
' Started....
' Updated.... 15 MAY 2005
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
' =====

' ----- [ Program Description ] -----

' ----- [ Revision History ] -----

' ----- [ I/O Definitions ] -----

FpuClk      PIN    15           ' FPU.3
FpuOut      PIN    14           ' FPU.2*
FpuIn       PIN    14           ' FPU.5*

' * When sharing the same IO pin a 1K resistor must be placed between
'   FPU.2 and FPU.5

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

#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
  RstTime   CON    250
#CASE BS2SX, BS2P, BS2PX
  RstTime   CON    625
#ENDSELECT

SELECTA     CON    $00      ' select A register
SELECTB     CON    $10      ' select B register
FWRITEA     CON    $20      ' select A and write float to register
FWRITEB     CON    $30      ' select B and write float to register
FREAD       CON    $40      ' read float from register
FSET        CON    $50      ' A = REG
LSET        CON    $50      ' A = REG

FADD        CON    $60      ' A = A + REG (float)
FSUB        CON    $70      ' A = A - REG (float)
FMUL        CON    $80      ' A = A * REG (float)
```

FDIV	CON	\$90	' A = A / REG (float)
LADD	CON	\$A0	' A = A + REG (long)
LSUB	CON	\$B0	' A = A - REG (long)
LMUL	CON	\$C0	' A = A * REG (long)
LDIV	CON	\$D0	' A = A / REG (long)
SQRT	CON	\$E0	' A = sqrt(A)
LOG	CON	\$E1	' A = ln(A)
LOG10	CON	\$E2	' A = log(A)
EXP	CON	\$E3	' A = e ** A
EXP10	CON	\$E4	' A = 10 ** A
FSIN	CON	\$E5	' A = sin(A) radians
FCOS	CON	\$E6	' A = cos(A) radians
FTAN	CON	\$E7	' A = tan(A) radians
FLOOR	CON	\$E8	' A = nearest integer <= A
CEIL	CON	\$E9	' A = nearest integer >= A
ROUND	CON	\$EA	' A = nearest integer to A
NEGATE	CON	\$EB	' A = -A
FABS	CON	\$EC	' A = A
INVERSE	CON	\$ED	' A = 1 / A
DEGREES	CON	\$EE	' A = A / (PI / 180) radians to degrees
RADIANS	CON	\$EF	' A = A * (PI / 180) degrees to radians
SYNC	CON	\$F0	' synchronization
FLOAT	CON	\$F1	' copy A to register 0 and float
FIX	CON	\$F2	' copy A to register 0 and fix
FCOMPARE	CON	\$F3	' compare A and B
LOADBYTE	CON	\$F4	' write signed byte to register 0, convert to float
LOADUBYTE	CON	\$F5	' write unsigned byte to register 0, convert to float
LOADWORD	CON	\$F6	' write signed word to register 0, convert to float
LOADUWORD	CON	\$F7	' write unsigned word to register 0, convert to float
READSTR	CON	\$F8	' read zero terminated string
ATOF	CON	\$F9	' convert ASCII to float, store in A
FTOA	CON	\$FA	' convert float to ASCII
ATOL	CON	\$FB	' convert ASCII to long, store in A
LTOA	CON	\$FC	' convert long to ASCII
FSTATUS	CON	\$FD	' get the status of A register
XOP	CON	\$FE	' extended opcode
NOP	CON	\$FF	' nop
FUNCTION	CON	\$00	' (XOP) user functions 0-15
READBYTE	CON	\$90	' (XOP) read low 8 bits of A (long)
READWORD	CON	\$91	' (XOP) read low 16 bits of A (long)
READLONG	CON	\$92	' (XOP) read 32-bit long value from A
READFLOAT	CON	\$93	' (XOP) read floating point value from A
LINCA	CON	\$94	' (XOP) A = A + 1 (long)
LINCB	CON	\$95	' (XOP) A = A + 1 (long)
LDECA	CON	\$96	' (XOP) A = A - 1 (long)
LDECB	CON	\$97	' (XOP) A = A - 1 (long)
LAND	CON	\$98	' (XOP) A = A AND B (long)
LOR	CON	\$99	' (XOP) A = A OR B (long)

Column #123: Getting Hot, Hot, Hot

LXOR	CON	\$9A	' (XOP) A = A XOR B (long)
LNOT	CON	\$9B	' (XOP) A = NOT A (long)
LTST	CON	\$9C	' (XOP) status of A AND B (long)
LSHIFT	CON	\$9D	' (XOP) shift A by B bits (long)
LWRITEA	CON	\$A0	' (XOP) select A and write long to register
LWRITEB	CON	\$B0	' (XOP) select B and write long to register
LREAD	CON	\$C0	' (XOP) read 32-bit long from register
LUDIV	CON	\$D0	' (XOP) A = A / REG (unsigned long)
POWER	CON	\$E0	' (XOP) A = A ** B
ROOT	CON	\$E1	' (XOP) A = the Bth root of A
FMIN	CON	\$E2	' (XOP) A = minimum of A and B
FMAX	CON	\$E3	' (XOP) A = maximum of A and B
FRACTION	CON	\$E4	' (XOP) load register 0 with the fractional part of A
ASIN	CON	\$E5	' (XOP) A = asin(A) radians
ACOS	CON	\$E6	' (XOP) A = acos(A) radians
ATAN	CON	\$E7	' (XOP) A = atan(A) radians
ATAN2	CON	\$E8	' (XOP) A = atan(A/B)
LCOMPARE	CON	\$E9	' (XOP) long compare A and B
LUCOMPARE	CON	\$EA	' (XOP) unsigned long compare A and B
LSTATUS	CON	\$EB	' (XOP) long status
LNEGATE	CON	\$EC	' (XOP) A = -A (long)
LABS	CON	\$ED	' (XOP) A = A (long)
LEFT	CON	\$EE	' (XOP) right parenthesis
RIGHT	CON	\$EF	' (XOP) left parenthesis
LOADZERO	CON	\$F0	' (XOP) load register 0 with zero
LOADONE	CON	\$F1	' (XOP) load register 0 with 1.0
LOADE	CON	\$F2	' (XOP) load register 0 with e
LOADPI	CON	\$F3	' (XOP) load register 0 with pi
LONGBYTE	CON	\$F4	' (XOP) write signed byte to register 0, convert to long
LONGUBYTE	CON	\$F5	' (XOP) write unsigned byte to register 0, convert to long
LONGWORD	CON	\$F6	' (XOP) write signed word to register 0, convert to long
LONGUWORD	CON	\$F7	' (XOP) write unsigned word to register 0, convert to long
IEEEMODE	CON	\$F8	' (XOP) set IEEE mode (default)
PICMODE	CON	\$F9	' (XOP) set PIC mode
CHECKSUM	CON	\$FA	' (XOP) calculate code checksum
BREAK	CON	\$FB	' (XOP) debug breakpoint
TRACEOFF	CON	\$FC	' (XOP) turn debug trace off
TRACEON	CON	\$FD	' (XOP) turn debug trace on
TRACESTR	CON	\$FE	' (XOP) send debug string to trace buffer
VERSION	CON	\$FF	' (XOP) get version string
SyncChar	CON	\$5C	
' ----- [Variables] -----			
status	VAR	Byte	
dByte	VAR	Byte	
opCode	VAR	Byte	
format	VAR	Byte	

```

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

Reset:
  DEBUG CLS
  GOSUB FPU_Reset                    ' kick things off
  IF (status = SyncChar) THEN      ' FPU installed?
    GOSUB Print_Version            ' yes, show version
    DEBUG " detected", CR
    PAUSE 250
  ELSE
    DEBUG "uM-FPU not detected", CR
  END
ENDIF

' ----[ Program Code ]-----

Main:

  END

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

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

FPU_Reset:
  LOW FpuClk                        ' initialize IO pins
  LOW FpuOut
  PULSOUT FpuClk, RstTime           ' send reset pulse
  PAUSE 8                           ' allow for FPU reset
  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [SYNC] ' send SYNC
  SHIF TIN FpuIn, FpuClk, MSBPRES, [status] ' return status
  RETURN

FPU_Wait:
  INPUT FpuIn                       ' setup to monitor pin
  DO : LOOP WHILE (FpuIn = 1)       ' wait until it drops
  RETURN

Print_Version:
  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, VERSION]

```

Column #123: Getting Hot, Hot, Hot

```
GOTO Print_String2

Print_Float:
  format = 0                                ' free format

Print_FloatFormat:
  opcode = FTOA                             ' convert FP to ASCII
  GOTO Print_String

Print_Long:
  format = 0                                ' free format

Print_LongFormat:
  opcode = LTOA                             ' convert long to ASCII

Print_String:
  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [opcode, format]

Print_String2:
  GOSUB Fpu_Wait                            ' let FPU get read
  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [READSTR] ' read string
  DO
    SHIF TIN FpuIn, FpuClk, MSBPRE, [dByte]   ' get a byte
    IF (dByte = 0) OR (dByte > 127) THEN EXIT ' if zero, we're done
    DEBUG dByte                               ' print the character
  LOOP
  RETURN
```

```

' =====
'
' File..... DS1620_Std.BS2
' Purpose.... Simple DS1620 Interface and Temperature Calculations
' Author..... Jon Williams -- Parallax, Inc.
' E-mail..... jwilliams@parallax.com
' Started....
' Updated.... 12 MAY 2004
'
' {$STAMP BS2}
' {$PBASIC 2.5}
' =====
'
' -----[ Program Description ]-----
'
' -----[ Revision History ]-----
'
' -----[ I/O Definitions ]-----
DsDQ          PIN    0          ' DS1620.1 (data I/O)
DsClk         PIN    1          ' DS1620.2
DsRst         PIN    2          ' DS1620.3
'
' -----[ Constants ]-----
RdTmp         CON    $AA          ' read temperature
WrHi          CON    $01          ' write TH (high temp)
WrLo          CON    $02          ' write TL (low temp)
RdHi          CON    $A1          ' read TH
RdLo          CON    $A2          ' read TL
StartC        CON    $EE          ' start conversion
StopC         CON    $22          ' stop conversion
WrCfg         CON    $0C          ' write config register
RdCfg         CON    $AC          ' read config register
DegSym        CON    186          ' degrees symbol
'
' -----[ Variables ]-----
tempIn        VAR    Word          ' raw temperature
sign          VAR    tempIn.BIT8   ' 1 = negative temperature
tC            VAR    Word          ' Celsius
tF            VAR    Word          ' Fahrenheit

```

Column #123: Getting Hot, Hot, Hot

```
' ----- [ EEPROM Data ]-----  
  
' ----- [ Initialization ]-----  
  
Setup:  
HIGH DsRst                ' alert DS1620  
SHIFTOUT DsDQ, DsClk, LSBFIRST, [WrCfg, %11] ' with CPU, one-shot mode  
LOW DsRst                 ' release DS1620  
PAUSE 10                  ' allow EEPROM write  
  
DEBUG CLS,  
    "DS1620 Standard", CR,  
    CR,  
    "tC... ", CR,  
    "tF... "  
  
' ----- [ Program Code ]-----  
  
Main:  
  GOSUB Get_HR_Temp  
  
Show_C:  
  DEBUG CRSRXY, 6, 2,  
    (tC.BIT15 * 13 + 32),  
    DEC (ABS tC / 10), ".", DEC1 (ABS tC),  
    DegSym, CLREOL  
  
Show_F:  
  DEBUG CRSRXY, 6, 3,  
    (tF.BIT15 * 13 + 32),  
    DEC (ABS tF / 10), ".", DEC1 (ABS tF),  
    DegSym, CLREOL  
  
  PAUSE 500  
  GOTO Main  
  END  
  
' ----- [ Subroutines ]-----  
  
Get_HR_Temp:  
HIGH DsRst  
SHIFTOUT DsDQ, DsClk, LSBFIRST, [StartC]    ' start conversion  
LOW DsRst  
DO  
  HIGH DsRst  
  SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdCfg]    ' read config register  
  SHIFTOIN DsDQ, DsClk, LSBPRE, [tempIn\8]  
  LOW DsRst
```

```
LOOP UNTIL (tempIn.BIT7 = 1)           ' wait until conversion done

HIGH DsRst
SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdTmp] ' read temp
SHIFTIN DsDQ, DsClk, LSBPRE, [tempIn\9]
LOW DsRst

IF (sign = 0) THEN                     ' temp C is positive
  tC = tempIn * 5                       ' convert to tenths
  tF = tC * 9 / 5 + 320                 ' convert to F
ELSE                                     ' temp C is negative
  tC = tempIn | $FF00 * 5               ' convert to negative tenths
  tF = 320 - ((ABS tC) * 9 / 5)         ' convert to F
ENDIF
RETURN
```

Column #123: Getting Hot, Hot, Hot

```

' =====
'
' File..... DS1620_HR.BS2
' Purpose... DS1620 Interface / Hi-Res Temperature Calculations
' Author.... Jon Williams -- Parallax, Inc.
' E-mail.... jwilliams@parallax.com
' Started...
' Updated... 30 JUL 2004
'
' {$STAMP BS2}
' {$PBASIC 2.5}
' =====

' ---- [ Program Description ]-----

' ---- [ Revision History ]-----

' ---- [ I/O Definitions ]-----

DsDQ          PIN    0          ' DS1620.1 (data I/O)
DsClk         PIN    1          ' DS1620.2
DsRst         PIN    2          ' DS1620.3

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

RdTmp         CON    $AA          ' read temperature
WrHi          CON    $01          ' write TH (high temp)
WrLo          CON    $02          ' write TL (low temp)
RdHi          CON    $A1          ' read TH
RdLo          CON    $A2          ' read TL
StartC        CON    $EE          ' start conversion
StopC         CON    $22          ' stop conversion
WrCfg         CON    $0C          ' write config register
RdCfg         CON    $AC          ' read config register
RdCnt         CON    $A0          ' read counter
RdSlope       CON    $A9          ' read slope

DegSym        CON    186          ' degrees symbol

' ---- [ Variables ]-----

tempIn        VAR    Word          ' raw temperature
sign          VAR    tempIn.BIT8   ' 1 = negative temperature
cRem          VAR    Word          ' count remaining
slope         VAR    Word          ' slope (counts per degree)

```

```

tC          VAR      Word          ' Celsius
tF          VAR      Word          ' Fahrenheit

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

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

Setup:
HIGH DsRst          ' alert DS1620
SHIFTOUT DsDQ, DsClk, LSBFIRST, [WrCfg, %11] ' with CPU, one-shot mode
LOW DsRst           ' release DS1620
PAUSE 10            ' allow EEPROM write

DEBUG CLS,
      "DS1620 Hi-Res", CR,
      CR,
      "tC... ", CR,
      "tF... "

' -----[ Program Code ]-----

Main:
GOSUB Get_HR_Temp

Show_C:
DEBUG CRSRXY, 6, 2,
      (tC.BIT15 * 13 + 32),
      DEC (ABS tC / 100), ".", DEC2 (ABS tC),
      DegSym, CLREOL

Show_F:
DEBUG CRSRXY, 6, 3,
      (tF.BIT15 * 13 + 32),
      DEC (ABS tF / 100), ".", DEC2 (ABS tF),
      DegSym, CLREOL

PAUSE 500
GOTO Main
END

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

Get_HR_Temp:
HIGH DsRst
SHIFTOUT DsDQ, DsClk, LSBFIRST, [StartC]      ' start conversion
LOW DsRst
DO

```

Column #123: Getting Hot, Hot, Hot

```
HIGH DsRst
  SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdCfg]      ' read config register
  SHIFTFIN DsDQ, DsClk, LSBPRE, [tempIn\8]
  LOW DsRst
LOOP UNTIL (tempIn.BIT7 = 1)                    ' wait until conversion done

HIGH DsRst
  SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdTmp]      ' read temp
  SHIFTFIN DsDQ, DsClk, LSBPRE, [tempIn\9]
  LOW DsRst

HIGH DsRst
  SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdCntr]     ' read counter
  SHIFTFIN DsDQ, DsClk, LSBPRE, [cRem\9]
  LOW DsRst

HIGH DsRst
  SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdSlope]    ' read slope
  SHIFTFIN DsDQ, DsClk, LSBPRE, [slope\9]
  LOW DsRst

tempIn = tempIn >> 1                          ' remove half degree bit
tempIn.BYTE1 = -tempIn.BIT7                    ' extend sign
tC = tempIn * 100                              ' convert to 100ths
tC = tC - 25 + (slope - cRem * 100 / slope)    ' fix fractional temp
IF (tC.BIT15 = 0) THEN
  tF = tC * 9 / 5 + 3200                       ' convert pos C to Fahr
ELSE
  tF = 3200 - ((ABS tC) * 9 / 5)              ' convert neg C to Fahr
ENDIF

RETURN
```

```

' =====
'
' File..... DS1620_FPU2.BS2
' Purpose.... DS1620 Interface / Hi-Res Calculations with FPU
' Author..... Jon Williams -- Parallax, Inc.
' E-mail..... jwilliams@parallax.com
' Started....
' Updated.... 15 MAY 2005
'
' {$STAMP BS2sx}
' {$PBASIC 2.5}
' =====

' ----[ I/O Definitions ]-----

DsDQ          PIN    0          ' DS1620.1 (data I/O)
DsClk         PIN    1          ' DS1620.2
DsRst         PIN    2          ' DS1620.3

FpuClk        PIN    15         ' FPU.3
FpuOut        PIN    14         ' FPU.2*
FpuIn         PIN    14         ' FPU.5*

' * When sharing the same IO pin a 1K resistor must be placed between
'   FPU.2 and FPU.5

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

#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
  RstTime     CON    250
#CASE BS2SX, BS2P, BS2FX
  RstTime     CON    625
#ENDSELECT

SELECTA       CON    $00      ' select A register
SELECTB       CON    $10      ' select B register
FWRITEA       CON    $20      ' select A and write float to register
FWRITEB       CON    $30      ' select B and write float to register
FREAD         CON    $40      ' read float from register
FSET          CON    $50      ' A = REG
LSET          CON    $50      ' A = REG

FADD          CON    $60      ' A = A + REG (float)
FSUB          CON    $70      ' A = A - REG (float)
FMUL          CON    $80      ' A = A * REG (float)
FDIV          CON    $90      ' A = A / REG (float)
LADD          CON    $A0      ' A = A + REG (long)

```

Column #123: Getting Hot, Hot, Hot

LSUB	CON	\$B0	' A = A - REG (long)
LMUL	CON	\$C0	' A = A * REG (long)
LDIV	CON	\$D0	' A = A / REG (long)
SQRT	CON	\$E0	' A = sqrt(A)
LOG	CON	\$E1	' A = ln(A)
LOG10	CON	\$E2	' A = log(A)
EXP	CON	\$E3	' A = e ** A
EXP10	CON	\$E4	' A = 10 ** A
FSIN	CON	\$E5	' A = sin(A) radians
FCOS	CON	\$E6	' A = cos(A) radians
FTAN	CON	\$E7	' A = tan(A) radians
FLOOR	CON	\$E8	' A = nearest integer <= A
CEIL	CON	\$E9	' A = nearest integer >= A
ROUND	CON	\$EA	' A = nearest integer to A
NEGATE	CON	\$EB	' A = -A
FABS	CON	\$EC	' A = A
INVERSE	CON	\$ED	' A = 1 / A
DEGREES	CON	\$EE	' A = A / (PI / 180) radians to degrees
RADIANS	CON	\$EF	' A = A * (PI / 180) degrees to radians
SYNC	CON	\$F0	' synchronization
FLOAT	CON	\$F1	' copy A to register 0 and float
FIX	CON	\$F2	' copy A to register 0 and fix
FCOMPARE	CON	\$F3	' compare A and B
LOADBYTE	CON	\$F4	' write signed byte to register 0, convert to float
LOADUBYTE	CON	\$F5	' write unsigned byte to register 0, convert to float
LOADWORD	CON	\$F6	' write signed word to register 0, convert to float
LOADUWORD	CON	\$F7	' write unsigned word to register 0, convert to float
READSTR	CON	\$F8	' read zero terminated string
ATOF	CON	\$F9	' convert ASCII to float, store in A
FTOA	CON	\$FA	' convert float to ASCII
ATOL	CON	\$FB	' convert ASCII to long, store in A
LTOA	CON	\$FC	' convert long to ASCII
FSTATUS	CON	\$FD	' get the status of A register
XOP	CON	\$FE	' extended opcode
NOP	CON	\$FF	' nop
FUNCTION	CON	\$00	' (XOP) user functions 0-15
READBYTE	CON	\$90	' (XOP) read low 8 bits of A (long)
READWORD	CON	\$91	' (XOP) read low 16 bits of A (long)
READLONG	CON	\$92	' (XOP) read 32-bit long value from A
READFLOAT	CON	\$93	' (XOP) read floating point value from A
LINCA	CON	\$94	' (XOP) A = A + 1 (long)
LINCB	CON	\$95	' (XOP) A = A + 1 (long)
LDECA	CON	\$96	' (XOP) A = A - 1 (long)
LDECB	CON	\$97	' (XOP) A = A - 1 (long)
LAND	CON	\$98	' (XOP) A = A AND B (long)
LOR	CON	\$99	' (XOP) A = A OR B (long)
LXOR	CON	\$9A	' (XOP) A = A XOR B (long)
LNOT	CON	\$9B	' (XOP) A = NOT A (long)
LTST	CON	\$9C	' (XOP) status of A AND B (long)

LSHIFT	CON	\$9D	' (XOP) shift A by B bits (long)
LWRITEA	CON	\$A0	' (XOP) select A and write long to register
LWRITEB	CON	\$B0	' (XOP) select B and write long to register
LREAD	CON	\$C0	' (XOP) read 32-bit long from register
LUDIV	CON	\$D0	' (XOP) A = A / REG (unsigned long)
POWER	CON	\$E0	' (XOP) A = A ** B
ROOT	CON	\$E1	' (XOP) A = the Bth root of A
FMIN	CON	\$E2	' (XOP) A = minimum of A and B
FMAX	CON	\$E3	' (XOP) A = maximum of A and B
FRACTION	CON	\$E4	' (XOP) load register 0 with the fractional part of A
ASIN	CON	\$E5	' (XOP) A = asin(A) radians
ACOS	CON	\$E6	' (XOP) A = acos(A) radians
ATAN	CON	\$E7	' (XOP) A = atan(A) radians
ATAN2	CON	\$E8	' (XOP) A = atan(A/B)
LCOMPARE	CON	\$E9	' (XOP) long compare A and B
LUCOMPARE	CON	\$EA	' (XOP) unsigned long compare A and B
LSTATUS	CON	\$EB	' (XOP) long status
LNEGATE	CON	\$EC	' (XOP) A = -A (long)
LABS	CON	\$ED	' (XOP) A = A (long)
LEFT	CON	\$EE	' (XOP) right parenthesis
RIGHT	CON	\$EF	' (XOP) left parenthesis
LOADZERO	CON	\$F0	' (XOP) load register 0 with zero
LOADONE	CON	\$F1	' (XOP) load register 0 with 1.0
LOADE	CON	\$F2	' (XOP) load register 0 with e
LOADPI	CON	\$F3	' (XOP) load register 0 with pi
LONGBYTE	CON	\$F4	' (XOP) write signed byte to register 0, convert to long
LONGUBYTE	CON	\$F5	' (XOP) write unsigned byte to register 0, convert to long
LONGWORD	CON	\$F6	' (XOP) write signed word to register 0, convert to long
LONGUWORD	CON	\$F7	' (XOP) write unsigned word to register 0, convert to long
IEEEMODE	CON	\$F8	' (XOP) set IEEE mode (default)
PICMODE	CON	\$F9	' (XOP) set PIC mode
CHECKSUM	CON	\$FA	' (XOP) calculate code checksum
BREAK	CON	\$FB	' (XOP) debug breakpoint
TRACEOFF	CON	\$FC	' (XOP) turn debug trace off
TRACEON	CON	\$FD	' (XOP) turn debug trace on
TRACESTR	CON	\$FE	' (XOP) send debug string to trace buffer
VERSION	CON	\$FF	' (XOP) get version string
SyncChar	CON	\$5C	
RdTmp	CON	\$AA	' read temperature
WrHi	CON	\$01	' write TH (high temp)
WrLo	CON	\$02	' write TL (low temp)
RdHi	CON	\$A1	' read TH
RdLo	CON	\$A2	' read TL
StartC	CON	\$EE	' start conversion
StopC	CON	\$22	' stop conversion
WrCfg	CON	\$0C	' write config register
RdCfg	CON	\$AC	' read config register
RdCntr	CON	\$A0	' read counter

Column #123: Getting Hot, Hot, Hot

```
RdSlope      CON    $A9          ' read slope
DegSym       CON    186          ' degrees symbol
' FPU registers
RawT         CON    1            ' uM-FPU register 1
Counts       CON    2            ' uM-FPU register 2
Slope        CON    3            ' uM-FPU register 3
TempC        CON    4            ' uM-FPU register 4
TempF        CON    5            ' uM-FPU register 5

' -----[ Variables ]-----

status       VAR    Byte
dByte        VAR    Byte
opCode       VAR    Byte
format       VAR    Byte

work         VAR    Word          ' data from DS1620
tC           VAR    Word          ' Celsius
tF           VAR    Word          ' Fahrenheit

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

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

Setup:
  DEBUG CLS
  GOSUB FPU_Reset                ' kick things off
  IF (status <> SyncChar) THEN   ' FPU installed?
    DEBUG "uM-FPU not detected", CR
  END
  ENDIF

  HIGH DsRst                    ' alert DS1620
  SHIFTOUT DsDQ, DsClk, LSBFIRST, [WrCfg, %11] ' with CPU, one-shot mode
  LOW DsRst                      ' release DS1620
  PAUSE 10                       ' allow EEPROM write

  DEBUG CLS,
    "DS1620 Hi-Res (FPU)", CR,
    CR,
    "tC... ", CR,
    "tF... "

' -----[ Program Code ]-----
```

```

Main:
  GOSUB Get_HR_Temp

Show_C:
  DEBUG CRSRXY, 6, 2,
    (tC.BIT15 * 13 + 32),
    DEC (ABS tC / 100), ".", DEC2 (ABS tC),
    DegSym, CLREOL

Show_F:
  DEBUG CRSRXY, 6, 3,
    (tF.BIT15 * 13 + 32),
    DEC (ABS tF / 100), ".", DEC2 (ABS tF),
    DegSym, CLREOL

  PAUSE 500
  GOTO Main
END

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

Get_HR_Temp:
  HIGH DsRst
  SHIFTOUT DsDQ, DsClk, LSBFIRST, [StartC]      ' start conversion
  LOW DsRst
  DO
    HIGH DsRst
    SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdCfg]      ' read config register
    SHIFTIM DsDQ, DsClk, LSBPRE, [work\8]
    LOW DsRst
  LOOP UNTIL (work.BIT7 = 1)                    ' wait until conversion done

  HIGH DsRst
  SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdTmp]      ' read temp
  SHIFTIM DsDQ, DsClk, LSBPRE, [work\9]
  LOW DsRst

  work = work >> 1                             ' remove half-bit
  IF (work.BIT7 = 1) THEN                       ' negative?
    work = work | $FF00                         ' if yes, extend sign bits
  ENDIF

  ' ----- RawT = work
  SHIFTOUT FpuOut, FpuClk, MSBFIRST,
    [RawT, LOADWORD, work.HIGHBYTE, work.LOWBYTE, FSET]

  HIGH DsRst
  SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdCntr]     ' read counter
  SHIFTIM DsDQ, DsClk, LSBPRE, [work\9]

```

Column #123: Getting Hot, Hot, Hot

```
LOW DsRst

' ----- Counts = work
SHIFTOUT FpuOut, FpuClk, MSBFIRST,
    [Counts, LOADWORD, work.HIGHBYTE, work.LOWBYTE, FSET]

HIGH DsRst
SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdSlope]      ' read stope
SHIFTIN DsDQ, DsClk, LSBPRE, [work\9]
LOW DsRst

' ----- Slope = work
SHIFTOUT FpuOut, FpuClk, MSBFIRST,
    [Slope, LOADWORD, work.HIGHBYTE, work.LOWBYTE, FSET]

' ----- TempC = RawT - 0.25 + ((Slope - Counts) / Slope)
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [TempC, FSET+RawT,
    FWRITEB, $BE, $80, $00, $00, FADD, XOP, LEFT,
    XOP, LEFT, FSET+Slope, FSUB+Counts, XOP, RIGHT,
    FSET, FDIV+Slope, XOP, RIGHT, FADD]

' ----- tC = ROUND (TempC * 100)
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [SELECTA]
GOSUB Fpu Wait
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, LEFT, XOP, LEFT,
    FSET+TempC, FWRITEB, $42, $C8, $00, $00, FMUL, ROUND,
    XOP, RIGHT, FSET, XOP, RIGHT, FIX]
GOSUB Fpu Wait
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, READWORD]
SHIFTIN FpuIn, FpuClk, MSBPRES, [tC.HIGHBYTE, tC.LOWBYTE]

' ----- TempF = TempC * 1.8 + 32.0
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [TempF, XOP, LEFT, FSET+TempC,
    FWRITEB, $3F, $E6, $66, $66, FMUL, XOP, RIGHT, FSET,
    FWRITEB, $42, $00, $00, $00, FADD]

' ----- tF = ROUND (TempF * 100)
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [SELECTA, XOP, LEFT, XOP,
    LEFT, FSET+TempF, FWRITEB, $42, $C8, $00, $00, FMUL,
    ROUND, XOP, RIGHT, FSET, XOP, RIGHT, FIX]
GOSUB Fpu Wait
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, READWORD]
SHIFTIN FpuIn, FpuClk, MSBPRES, [tF.HIGHBYTE, tF.LOWBYTE]
RETURN

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

FPU_Reset:
    LOW FpuClk                ' initialize IO pins
    LOW FpuOut
```

```

PULSOUT FpuClk, RstTime           ' send reset pulse
PAUSE 8                           ' allow for FPU reset
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [SYNC] ' send SYNC
SHIFTIN FpuIn, FpuClk, MSBPRES, [status] ' return status
RETURN

FPU Wait:
  INPUT FpuIn                       ' setup to monitor pin
  DO : LOOP WHILE (FpuIn = 1)       ' wait until it drops
  RETURN

Print_Version:
  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, VERSION]
  GOTO Print_String2

Print_Float:
  format = 0                         ' free format

Print_FloatFormat:
  opcode = FTOA                      ' convert FP to ASCII
  GOTO Print_String

Print_Long:
  format = 0                         ' free format

Print_LongFormat:
  opcode = LTOA                      ' convert long to ASCII

Print_String:
  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [opcode, format]

Print_String2:
  GOSUB Fpu_Wait                     ' let FPU get read
  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [READSTR] ' read string
  DO
    SHIFTIN FpuIn, FpuClk, MSBPRES, [dByte] ' get a byte
    IF (dByte = 0) OR (dByte > 127) THEN EXIT ' if zero, we're done
    DEBUG dByte                       ' print the character
  LOOP
  RETURN

```

Column #123: Getting Hot, Hot, Hot

```
' =====
'
' File..... DS1620_FPU2.BS2
' Purpose... DS1620 Interface / Hi-Res Calculations with FPU
' Author.... Jon Williams -- Parallax, Inc.
' E-mail.... jwilliams@parallax.com
' Started...
' Updated... 15 MAY 2005
'
'   {$STAMP BS2sx}
'   {$PBASIC 2.5}
'
' =====

' ----[ Program Description ]-----
'
' This version of the program has the calculations embedded in the FPU.

' ----[ Revision History ]-----

' ----[ I/O Definitions ]-----

DsDQ      PIN    0           ' DS1620.1 (data I/O)
DsClk     PIN    1           ' DS1620.2
DsRst     PIN    2           ' DS1620.3

FpuClk    PIN    15          ' FPU.3
FpuOut    PIN    14          ' FPU.2*
FpuIn     PIN    14          ' FPU.5*

' * When sharing the same IO pin a 1K resistor must be placed between
'   FPU.2 and FPU.5

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

#SELECT $STAMP
#CASE BS2, BS2E, BS2PE
  RstTime  CON    250
#CASE BS2SX, BS2P, BS2PX
  RstTime  CON    625
#ENDSELECT

SELECTA   CON    $00        ' select A register
SELECTB   CON    $10        ' select B register
FWRITEA   CON    $20        ' select A and write float to register
FWRITEB   CON    $30        ' select B and write float to register
FREAD     CON    $40        ' read float from register
```

FSET	CON	\$50	' A = REG
LSET	CON	\$50	' A = REG
FADD	CON	\$60	' A = A + REG (float)
FSUB	CON	\$70	' A = A - REG (float)
FMUL	CON	\$80	' A = A * REG (float)
FDIV	CON	\$90	' A = A / REG (float)
LADD	CON	\$A0	' A = A + REG (long)
LSUB	CON	\$B0	' A = A - REG (long)
LMUL	CON	\$C0	' A = A * REG (long)
LDIV	CON	\$D0	' A = A / REG (long)
SQRT	CON	\$E0	' A = sqrt(A)
LOG	CON	\$E1	' A = ln(A)
LOG10	CON	\$E2	' A = log(A)
EXP	CON	\$E3	' A = e ** A
EXP10	CON	\$E4	' A = 10 ** A
FSIN	CON	\$E5	' A = sin(A) radians
FCOS	CON	\$E6	' A = cos(A) radians
FTAN	CON	\$E7	' A = tan(A) radians
FLOOR	CON	\$E8	' A = nearest integer <= A
CEIL	CON	\$E9	' A = nearest integer >= A
ROUND	CON	\$EA	' A = nearest integer to A
NEGATE	CON	\$EB	' A = -A
FABS	CON	\$EC	' A = A
INVERSE	CON	\$ED	' A = 1 / A
DEGREES	CON	\$EE	' A = A / (PI / 180) radians to degrees
RADIANS	CON	\$EF	' A = A * (PI / 180) degrees to radians
SYNC	CON	\$F0	' synchronization
FLOAT	CON	\$F1	' copy A to register 0 and float
FIX	CON	\$F2	' copy A to register 0 and fix
FCOMPARE	CON	\$F3	' compare A and B
LOADBYTE	CON	\$F4	' write signed byte to register 0, convert to float
LOADUBYTE	CON	\$F5	' write unsigned byte to register 0, convert to float
LOADWORD	CON	\$F6	' write signed word to register 0, convert to float
LOADUWORD	CON	\$F7	' write unsigned word to register 0, convert to float
READSTR	CON	\$F8	' read zero terminated string
ATOF	CON	\$F9	' convert ASCII to float, store in A
FTOA	CON	\$FA	' convert float to ASCII
ATOL	CON	\$FB	' convert ASCII to long, store in A
LTOA	CON	\$FC	' convert long to ASCII
FSTATUS	CON	\$FD	' get the status of A register
XOP	CON	\$FE	' extended opcode
NOP	CON	\$FF	' nop
FUNCTION	CON	\$00	' (XOP) user functions 0-15
READBYTE	CON	\$90	' (XOP) read low 8 bits of A (long)
READWORD	CON	\$91	' (XOP) read low 16 bits of A (long)
READLONG	CON	\$92	' (XOP) read 32-bit long value from A
READFLOAT	CON	\$93	' (XOP) read floating point value from A

Column #123: Getting Hot, Hot, Hot

LINCA	CON	\$94	' (XOP) A = A + 1 (long)
LINCB	CON	\$95	' (XOP) A = A + 1 (long)
LDECA	CON	\$96	' (XOP) A = A - 1 (long)
LDECB	CON	\$97	' (XOP) A = A - 1 (long)
LAND	CON	\$98	' (XOP) A = A AND B (long)
LOR	CON	\$99	' (XOP) A = A OR B (long)
LXOR	CON	\$9A	' (XOP) A = A XOR B (long)
LNOT	CON	\$9B	' (XOP) A = NOT A (long)
LTST	CON	\$9C	' (XOP) status of A AND B (long)
LSHIFT	CON	\$9D	' (XOP) shift A by B bits (long)
LWRITEA	CON	\$A0	' (XOP) select A and write long to register
LWRITEB	CON	\$B0	' (XOP) select B and write long to register
LREAD	CON	\$C0	' (XOP) read 32-bit long from register
LUDIV	CON	\$D0	' (XOP) A = A / REG (unsigned long)
POWER	CON	\$E0	' (XOP) A = A ** B
ROOT	CON	\$E1	' (XOP) A = the Bth root of A
FMIN	CON	\$E2	' (XOP) A = minimum of A and B
FMAX	CON	\$E3	' (XOP) A = maximum of A and B
FRACTION	CON	\$E4	' (XOP) load register 0 with the fractional part of A
ASIN	CON	\$E5	' (XOP) A = asin(A) radians
ACOS	CON	\$E6	' (XOP) A = acos(A) radians
ATAN	CON	\$E7	' (XOP) A = atan(A) radians
ATAN2	CON	\$E8	' (XOP) A = atan(A/B)
LCOMPARE	CON	\$E9	' (XOP) long compare A and B
LUCOMPARE	CON	\$EA	' (XOP) unsigned long compare A and B
LSTATUS	CON	\$EB	' (XOP) long status
LNEGATE	CON	\$EC	' (XOP) A = -A (long)
LABS	CON	\$ED	' (XOP) A = A (long)
LEFT	CON	\$EE	' (XOP) right parenthesis
RIGHT	CON	\$EF	' (XOP) left parenthesis
LOADZERO	CON	\$F0	' (XOP) load register 0 with zero
LOADONE	CON	\$F1	' (XOP) load register 0 with 1.0
LOADE	CON	\$F2	' (XOP) load register 0 with e
LOADPI	CON	\$F3	' (XOP) load register 0 with pi
LONGBYTE	CON	\$F4	' (XOP) write signed byte to register 0, convert to long
LONGUBYTE	CON	\$F5	' (XOP) write unsigned byte to register 0, convert to long
LONGWORD	CON	\$F6	' (XOP) write signed word to register 0, convert to long
LONGUWORD	CON	\$F7	' (XOP) write unsigned word to register 0, convert to long
IEEEMODE	CON	\$F8	' (XOP) set IEEE mode (default)
PICMODE	CON	\$F9	' (XOP) set PIC mode
CHECKSUM	CON	\$FA	' (XOP) calculate code checksum
BREAK	CON	\$FB	' (XOP) debug breakpoint
TRACEOFF	CON	\$FC	' (XOP) turn debug trace off
TRACEON	CON	\$FD	' (XOP) turn debug trace on
TRACESTR	CON	\$FE	' (XOP) send debug string to trace buffer
VERSION	CON	\$FF	' (XOP) get version string
SyncChar	CON	\$5C	
RdTmp	CON	\$AA	' read temperature

```

WrHi          CON    $01          ' write TH (high temp)
WrLo          CON    $02          ' write TL (low temp)
RdHi          CON    $A1          ' read TH
RdLo          CON    $A2          ' read TL
StartC        CON    $EE          ' start conversion
StopC         CON    $22          ' stop conversion
WrCfg         CON    $0C          ' write config register
RdCfg         CON    $AC          ' read config register
RdCntR        CON    $A0          ' read counter
RdSlope       CON    $A9          ' read slope

DegSym        CON    186          ' degrees symbol

' FPU registers

RawT          CON    1            ' uM-FPU register 1
Counts        CON    2            ' uM-FPU register 2
Slope         CON    3            ' uM-FPU register 3
TempC         CON    4            ' uM-FPU register 4
TempF         CON    5            ' uM-FPU register 5
TempCL        CON    6            ' uM-FPU register 6
TempFL        CON    7            ' uM-FPU register 7

' -----[ Variables ]-----

status        VAR    Byte
dByte         VAR    Byte
opCode        VAR    Byte
format        VAR    Byte

work          VAR    Word          ' data to/from DS1620

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

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

Setup:
  DEBUG CLS
  GOSUB FPU_Reset                    ' kick things off
  IF (status <> SyncChar) THEN      ' FPU installed?
    DEBUG "uM-FPU not detected", CR
  END
ENDIF

HIGH DsRst                    ' alert DS1620
SHIFTOUT DsDQ, DsClk, LSBFIRST, [WrCfg, %11] ' with CPU, one-shot mode
LOW DsRst                      ' release DS1620
PAUSE 10                       ' allow EEPROM write

```

Column #123: Getting Hot, Hot, Hot

```
DEBUG CLS,
      "DS1620 Hi-Res (FPU)", CR,
      CR,
      "tC... ", CR,
      "tF... "

' -----[ Program Code ]-----

Main:
  GOSUB Get_HR_Temp

Show_C:
  ' ----- @Calc_TC
  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, FUNCTION]

  ' ----- work = TempCL
  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [TempCL]
  GOSUB Fpu_Wait
  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, READWORD]
  SHIFTIIN FpuIn, FpuClk, MSBPRES, [work.HIGHBYTE, work.LOWBYTE]

  DEBUG CRSRXY, 6, 2,
        (work.BIT15 * 13 + 32),
        DEC (ABS work / 100), ".", DEC2 (ABS work),
        DegSym, CLREOL

Show_F:
  ' ----- @Calc_TF
  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, FUNCTION+1]

  ' ----- work = TempFL
  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [TempFL]
  GOSUB Fpu_Wait
  SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, READWORD]
  SHIFTIIN FpuIn, FpuClk, MSBPRES, [work.HIGHBYTE, work.LOWBYTE]

  DEBUG CRSRXY, 6, 3,
        (work.BIT15 * 13 + 32),
        DEC (ABS work / 100), ".", DEC2 (ABS work),
        DegSym, CLREOL

  PAUSE 500
  GOTO Main
  END

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

Get_HR_Temp:
```

```

HIGH DsRst
SHIFTOUT DsDQ, DsClk, LSBFIRST, [StartC]      ' start conversion
LOW DsRst
DO
  HIGH DsRst
  SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdCfg]      ' read config register
  SHIFTOIN DsDQ, DsClk, LSBPRE, [work\8]
  LOW DsRst
LOOP UNTIL (work.BIT7 = 1)                    ' wait until conversion done

HIGH DsRst
SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdTmp]      ' read temp
SHIFTOIN DsDQ, DsClk, LSBPRE, [work\9]
LOW DsRst

work = work >> 1                             ' remove half-bit
IF (work.BIT7 = 1) THEN                       ' negative?
  work = work | $FF00                         ' if yes, extend sign bits
ENDIF

' ----- RawT = work
SHIFTOUT FpuOut, FpuClk, MSBFIRST,
  [RawT, LOADWORD, work.HIGHBYTE, work.LOWBYTE, FSET]

HIGH DsRst
SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdCntr]      ' read counter
SHIFTOIN DsDQ, DsClk, LSBPRE, [work\9]
LOW DsRst

' ----- Counts = work
SHIFTOUT FpuOut, FpuClk, MSBFIRST,
  [Counts, LOADWORD, work.HIGHBYTE, work.LOWBYTE, FSET]

HIGH DsRst
SHIFTOUT DsDQ, DsClk, LSBFIRST, [RdSlope]     ' read slope
SHIFTOIN DsDQ, DsClk, LSBPRE, [work\9]
LOW DsRst

' ----- Slope = work
SHIFTOUT FpuOut, FpuClk, MSBFIRST,
  [Slope, LOADWORD, work.HIGHBYTE, work.LOWBYTE, FSET]

RETURN

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

FPU_Reset:
LOW FpuClk                                   ' initialize IO pins
LOW FpuOut
PULSOUT FpuClk, RstTime                     ' send reset pulse

```

Column #123: Getting Hot, Hot, Hot

```
PAUSE 8 ' allow for FPU reset
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [SYNC] ' send SYNC
SHIFTIN FpuIn, FpuClk, MSBPRES, [status] ' return status
RETURN

FPU_Wait:
INPUT FpuIn ' setup to monitor pin
DO : LOOP WHILE (FpuIn = 1) ' wait until it drops
RETURN

Print_Version:
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [XOP, VERSION]
GOTO Print_String2

Print_Float:
format = 0 ' free format

Print_FloatFormat:
opcode = FTOA ' convert FP to ASCII
GOTO Print_String

Print_Long:
format = 0 ' free format

Print_LongFormat:
opcode = LTOA ' convert long to ASCII

Print_String:
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [opcode, format]

Print_String2:
GOSUB Fpu_Wait ' let FPU get read
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [READSTR] ' read string
DO
  SHIFTIN FpuIn, FpuClk, MSBPRES, [dByte] ' get a byte
  IF (dByte = 0) OR (dByte > 127) THEN EXIT ' if zero, we're done
  DEBUG dByte ' print the character
LOOP
RETURN
```