Column #81, January 2002 by Jon Williams:

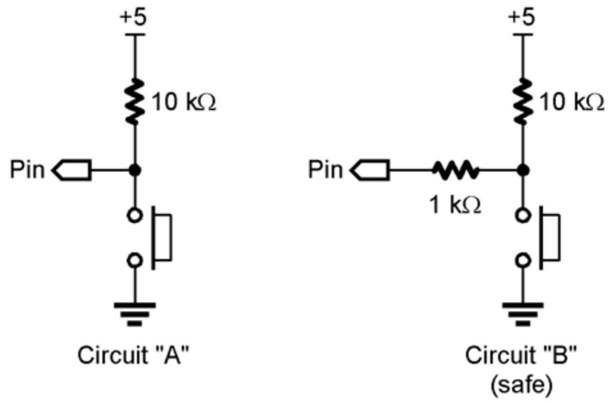
A Tale of Two Stamps

A friend of mine recently asked me how I come up with project ideas for this column. I thought for a second, then replied, "I don't ... I just try to pay attention to the e-mail I've been getting lately." Well, lately I've been getting a lot of e-mail on connecting things. Specifically, connecting two Stamps together. Another common question is how to connect a DS1302 and DS1620 together with the fewest number of pins. So that's what we're going to do. It's a simple project this month, but has some useful tips and techniques for your Stamp projects.

This article is inspired in large part by a young college student in Florida. His background is in computer science, not electronics. The connections thing is what got him in trouble. He called me one day, a bit dismayed, that he had connected two Stamps together and had done everything "just like it says in the manual" and that one of his Stamps "blew up."

I told him I was quite sure that he didn't do it "just like in the manual" otherwise there would be no dead Stamp. But that's little comfort to a student on a student's budget – BASIC Stamps aren't free. He wanted to know what he should do to fix his problem and to prevent it from happening again. I told him to relax, that there is an easy answer and not too feel badly; every engineer worth his or her salt has a collection of dead parts on their bench – yours truly included.

Figure 81.1: Pushbutton Circuit



The answer is simple. Just one word; three little syllables: resistor. I was giving a BASIC Stamp programming class to some electronics teachers last summer and I pointed to a component drawn on the white board. "What is this part?" I asked. "A resistor," they all obediently responded. "No it's not," I shot back. "That, my good friends, is cheap insurance."

The purpose of a resistor is to control or limit the flow of current. Since excess current is the killer of circuits (and people, for that matter), controlling current flow to and from the Stamp (I/O pins) will protect it. We can do this for less than a nickel with a resistor.

Take a look at Figure 81.1. Circuit "A" is a very typical push-button input. The input pin is pulled up to Vdd (+5) through a 10 KΩ resistor. When the button is open, the input will read a "1." When the button is pressed, the input will read as "0." This circuit presents no problem as long as the I/O pin is always an input. What can happen, however, is a programming error that causes the pin to become an output and go high. If this happens and the button is pushed, there is a direct short-circuit between the I/O pin and Vss (ground). Poof. Pin dead. Sometimes Stamp dead. Tarzan not happy.

If we spend a few cents and insert a 1 kΩ resistor in series with the pin and the switch – as in Circuit "B" – the same programming error does not kill the pin or the Stamp. The series resistor will limit the current flow to about five milliamps. This is well under the maximum current rating for an I/O pin.

If you're new at this, you may wonder what resistor values to have on hand. I'm about to tell you. These values will get you through 99% of your Stamp project requirements. The only time you'll need another value is if you're doing something very specific, as in an RC circuit.

220 Ω (use with RCTIME circuits)
470 Ω (use with LEDs)
1 kΩ (pin protection or with low-current LEDs)
4.7 kΩ (1-Wire and I^2C bus pull-ups)
10 kΩ (N.O. pushbutton pull-up/pull-downs)

Go to Radio Shack or your favorite electronics store and buy a bunch (at least 10) of each and keep them handy. I kid you not, I have a little five-compartment plastic box on my bench that holds these values. I rarely need anything else.

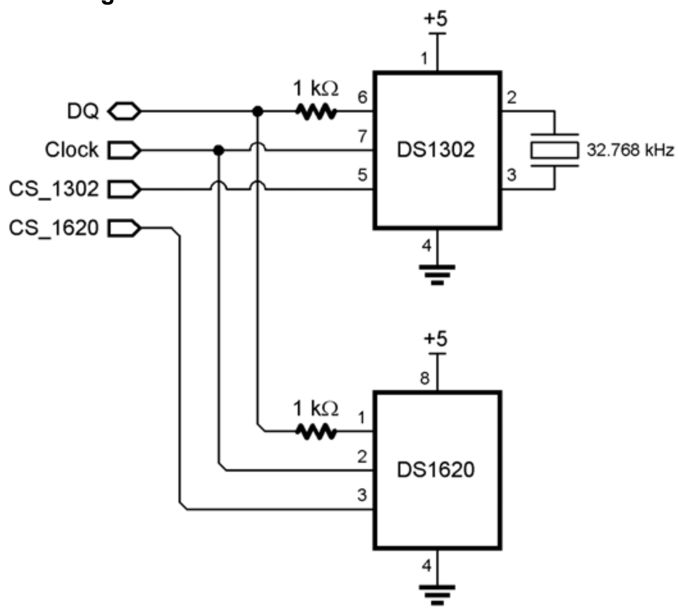
Talking Back And Forth

Our project this month is a very simple one and uses parts that are easy to get and that you may already have. We'll be using a lot of familiar code to keep things simple too. What we're going to do – as my young friend was trying to do – is connect two Stamps together in a Master-Slave arrangement.

We'll start with the slave, which really is acting more like a co-processor. Its responsibility is to manage the DS1302 real-time-clock, a DS1620 temperature sensor and a standard parallel LCD. Since we've used these parts so many times, I'm not going to go into the programming of those components. If you're a new reader, you can download StampWorks experiments from the Parallax web site that explain all of these components. Another good source of information on these components is "The Nuts & Volts of BASIC Stamps" book set, available from Nuts & Volts or Parallax.

The only thing that I do want to point out is how the DS1302 and the DS1620 share a couple of Stamp pins – a question that has come up quite frequently since Parallax published StampWorks. Figure 82.2 shows the DS1302 and DS1620 connections to the slave. Notice that each of the devices is connected to a pin generically called DQ (we can connect this to any available Stamp pin). Notice too, that each of the devices connects to that pin through a 1 kΩ resistor. The reason we do this is that the DQ pin on the DS1302 and DS1620 is bi-directional – it can be an input or an output, depending on what is going on at the moment.

Figure 81.2: DS1302 and DS1620 Serial Circuit



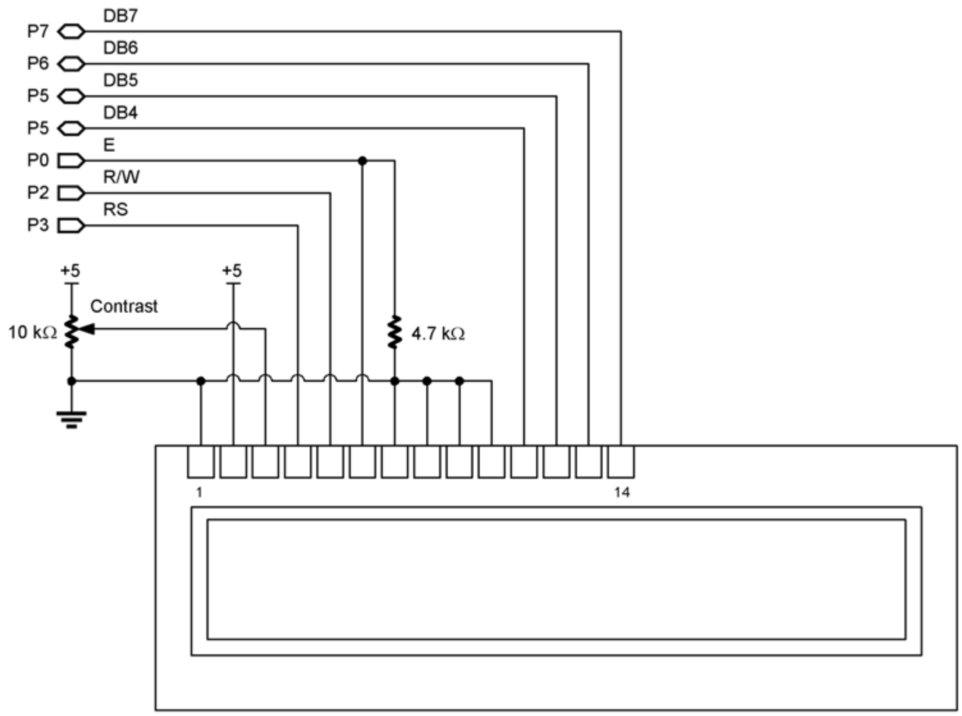
The resistor, as we saw in the push-button example above, will protect the Stamp in the event that the Stamp pin and the device pin both become outputs and of opposite polarity. If this occurs, there will be a little current flow, but no damage will be done. We don't need resistors on the clock (also shared) and device-select pins because those are inputs on the devices and are not bi-directional.

Figure 81.3 shows the LCD connections to the slave. This is the same was we used last month and is compatible with BS2p LCD commands – in case you want to use a BS2p as a co-processor with local display.

The slave Stamp program overview looks something like this:

Check for command from Master
    If command
     -- process and respond
    If no command
     -- get time and temp and update the LCD

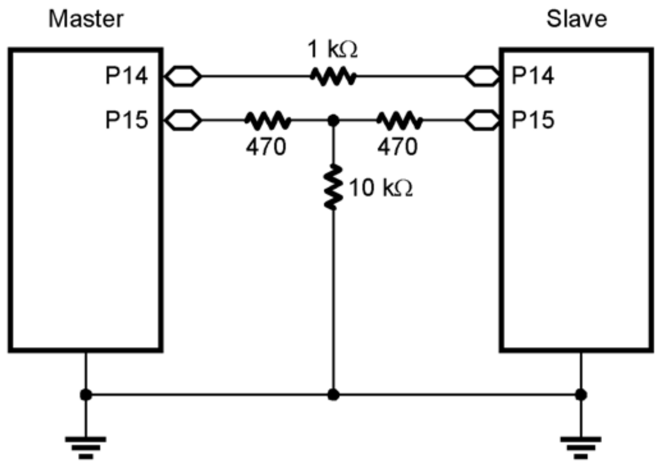
Figure 81.3: LCD Slave Connections



A lot of newbies get wrapped around the axle because the Stamp doesn't buffer serial communications. The fact is that most microcontrollers don't, and to do it requires a lot of resources. Thankfully, the Stamp **SERIN** command does have the ability to do flow control and can timeout if no data arrives. Combining flow control with timeout gives us an effective way to let the slave do its local work when no input from the master is available.

With flow control, the Stamp uses a second pin to tell the outside world that it is ready to receive serial input. In Stamp-to-Stamp communications, we'll use flow control with **SEROUT** as well. This will cause the transmitting Stamp NOT to send anything until the flow control signal from the receiver is present.

Figure 81.4: Master / Slave Connections



In case of the slave Stamp, we have a local display (LCD) the requires frequent updating. By using the timeout function of **SERIN**, we can abandon waiting for input from the master, update the display, then go back to waiting.

Program Listing 81.1 is the slave program. In the Main section of code, you'll see this line:

```
SERIN SIOpin\FCpin, N2400, 200, LCD_Update, [ioByte]
```

Notice the syntax change that specifies the flow control pin. A timeout value of 200 milliseconds means that, considering program overhead, the LCD is being updated about five times each second. This is enough frequency to keep the seconds display of the clock from getting jittery.

Since the program uses inverted serial data, the flow control pin will go high when **SERIN** is ready to receive. Figure 81.4 shows the serial connection between the two Stamps. Pin 14 on each Stamp is used for serial data, hence the 1 kΩ resistor between them. Pin 15 on each Stamp is used for flow control. The normal 1 kΩ resistance is divided (two 470 Ω resistors are close enough) and the 10 kΩ resistor will pull the inputs low, creating a "no go" signal for the sender when the receiver is initializing.

One more very important note: the two Stamps must have a common ground connection for this to work correctly. I had a bad connection while putting this project together and was getting all kinds of non-repeatable results. Once I discovered the errant ground connection, all problems went away and communication between the two Stamps proceeded easily.

Now that our slave Stamp is up and operating, let's give it a simple master to try-out this communications method. Program Listing 81.2 is the master program. It's very straightforward, simply displaying a menu with the time and temperature information from the slave. Using the menu, we can set the slave's clock and change the temperature display mode.

The master program waits for a single button press the converts the input (if valid) to a numeric command to send to the slave. **LOOKDOWN** is used to convert valid upper- and lower-case inputs to the proper command value. When "T" is pressed (to set the time) the program jumps to a bit of code that allows us to input the new hours and minutes for the clock. This section of code also uses serial timeouts so that the time display on the menu doesn't become invalid.

The master program is designed to work with an external terminal program. If you'd like to use the **DEBUG** window in the Stamp editor, just add a **DEBUG** line (anything will do) at the beginning of the program to open the window – the specified baud rate for the terminal matches **DEBUG**.

You see, it is very easy to connect a couple of Stamps, and other devices if you desire, and to do so in a manner that will prevent programming errors from creating hardware problems (failures). A common thread in the messages and calls for help that I've been getting lately is impatience. Even though I re-used a lot of code for this project, it still took a full day to put together and get working properly. Plan your projects and have patience. It will all pay off in the end.

Oops...

For those of you who thought last month's article seemed a bit short and as if it got cut-off, well, that's my fault. Silly me, I accidentally created two versions of the article file and I sent the wrong one to my friends at Nuts & Volts. I didn't even realize it until I saw the December article in print.

You can download the complete text from the Parallax web site. I'm sorry for the inconvenience and promise to be more careful in the future since a resistor can't prevent me from sending the wrong file.

Happy New Year and Happy Stamping.

```
' -----[ Title ]-----------------------------------------------------------
'
' Program Listing 81.1
' File...... MASTER.BS2
' Purpose... Master Stamp Demo -- manages terminal, uses Slave Stamp
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started...
' Updated... 08 DEC 2001

' { $STAMP BS2 }


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


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


' -----[ I/O Definitions ]-------------------------------------------------
'
SIOpin          CON     14                      ' serial I/O
FCpin           CON     15                      ' serial flow control
Tpin            CON     16                      ' terminal I/O


' -----[ Constants ]-------------------------------------------------------
'
N2400           CON     16780                   ' 2400-N81 inverted
T9600           CON     84                      ' 9600-N81 true (matches DEBUG)

Cmd_SetClock    CON     1                       ' commands for slave
Cmd_SetC        CON     2
Cmd_SetF        CON     3
Cmd_GetTmTmp    CON     4


' -----[ Variables ]-------------------------------------------------------
'
inByte          VAR     Byte                    ' input from user
cmd             VAR     Byte                    ' user command
hours           VAR     Byte                    ' clock hours (BCD)
minutes         VAR     Byte                    ' clock minutes (BCD)
temp            VAR     Word                    ' measured by slave
```

```
tMode           VAR     Byte                    ' "C" or "F"
digits          VAR     Nib                     ' for right justification


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


' -----[ Initialization ]--------------------------------------------------
'
Initialize:
  tMode = "C"                                   ' assume C until confirmed
  SEROUT Tpin, T9600, [CLS]                     ' clear terminal window
  SEROUT Tpin, T9600, ["Connecting..."]


' -----[ Main Code ]-------------------------------------------------------
'
Main:
  GOSUB Get_TimeTemp                            ' get data from Slave
  GOSUB Show_Menu                               ' display updated menu

Get_Input:
  SERIN Tpin, T9600, 5000, Main, [inByte]       ' wait 5 seconds for command
  cmd = 99
  LOOKDOWN inByte, ["tTcCfFrR"], cmd            ' convert letter to number
  cmd = (cmd / 2) + 1
  BRANCH cmd, [Main, Set_Time, Set_TMode, Set_TMode, Refresh]
  GOTO Main

Set_Time:
  SEROUT Tpin, T9600, [CR, CR, "Hours   (0..23) --> "]
  SERIN  Tpin, T9600, 5000, Refresh, [HEX3 hours]
  hours = hours // $23
  SEROUT Tpin, T9600, [CR, "Minutes (0..59) --> "]
  SERIN  Tpin, T9600, 5000, Refresh, [HEX3 minutes]
  minutes = minutes // $59
  SEROUT SIOpin\FCpin, N2400, [Cmd_SetClock]
  PAUSE 5
  SEROUT SIOpin\FCpin, N2400, [hours, minutes]
  GOTO Refresh

Set_TMode:
  SEROUT SIOpin\FCpin, N2400, [cmd]             ' send new temperature mode
  GOTO Main

Refresh:
  DEBUG CLS                                     ' clear terminal before menu
  GOTO Main                                     '   update
```

```
' -----[ Subroutines ]----------------------------------------------------------
'
Get_TimeTemp:
  SEROUT SIOpin\FCpin, N2400, [Cmd_GetTmTmp]
  SERIN  SIOpin\FCpin, N2400, 500, Get_TTX, [hours, minutes]
  SERIN  SIOpin\FCpin, N2400, 500, Get_TTX, [temp.LowByte, temp.HighByte, tMode]

Get_TTX:
  RETURN


Show_Menu:
  SEROUT Tpin, T9600, [Home]
  SEROUT Tpin, T9600, ["=======================", CR]

  SEROUT Tpin, T9600, [HEX2 hours, ":", HEX2 minutes]
  SEROUT Tpin, T9600, ["              "]
  digits = 3
  LOOKDOWN temp, <[0, 10, 100, 1000], digits
  SEROUT Tpin, T9600, [REP " "\(3-digits), DEC temp]    ' right justify temp
  SEROUT Tpin, T9600, ["°", tMode, CR]

  SEROUT Tpin, T9600, ["=======================", CR]
  SEROUT Tpin, T9600, ["[T] Set Time", CR]
  SEROUT Tpin, T9600, ["[C] Celcius Mode", CR]
  SEROUT Tpin, T9600, ["[F] Fahrenheit Mode", CR]
  SEROUT Tpin, T9600, ["[R] Refresh Screen", CR, CR]
  SEROUT Tpin, T9600, ["-->  ", BkSp]
  RETURN
```

```
' -----[ Title ]-----------------------------------------------------------
' Program Listing 81.2
' File...... SLAVE.BS2
' Purpose... Slave Stamp demo -- manages RTC, themometer and LCD
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' Started...
' Updated... 07 DEC 2001

' { $STAMP BS2 }


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


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


' -----[ I/O Definitions ]-------------------------------------------------
'
E               CON     0                       ' LCD Enable pin  (1 = enabled)
RS              CON     3                       ' Register Select (1 = char)
LCDout          VAR     OutB                    ' 4-bit LCD data

DQ              CON     8                       ' data line
Clock           CON     9
CS_1620         CON     10                      ' select DS1620 thermometer
CS_1302         CON     11                      ' select DS1302 RTC

SIOpin          CON     14                      ' serial I/O
FCpin           CON     15                      ' serial flow control


' -----[ Constants ]-------------------------------------------------------
'
N2400           CON     16780                   ' serial baud parameter

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

TM_Cels         CON     0                       ' temperature modes
```

```
TM_Fahr         CON     1

WrSecs          CON     $80                     ' write seconds
RdSecs          CON     $81                     ' read seconds
WrMins          CON     $82                     ' write minutes
RdMins          CON     $83                     ' read minutes
WrHrs           CON     $84                     ' write hours
RdHrs           CON     $85                     ' read hours
CWPr            CON     $8E                     ' write protect register
WPr1            CON     $80                     ' set write protect
WPr0            CON     $00                     ' clear write protect
WrBurst         CON     $BE                     ' write burst of data
RdBurst         CON     $BF                     ' read burst of data
WrRam           CON     $C0                     ' RAM address control
RdRam           CON     $C1

ClrLCD          CON     $01                     ' clear the LCD
CrsrHm          CON     $02                     ' move cursor to home position
CrsrLf          CON     $10                     ' move cursor left
CrsrRt          CON     $14                     ' move cursor right
DispLf          CON     $18                     ' shift displayed chars left
DispRt          CON     $1C                     ' shift displayed chars right
DDRam           CON     $80                     ' Display Data RAM control

NumTasks        CON     2                       ' Time and Temp

Tsk_Time        CON     0                       ' task control values
Tsk_Temp        CON     1


' -----[ Variables ]------------------------------------------------------------
'
task            VAR     Nib                     ' current task to execute
ioByte          VAR     Byte
rxBuffer        VAR     Byte(2)                 ' serial receive buffer
reg             VAR     Byte                    ' RTC register #

secs            VAR     Byte                    ' seconds
secs01          VAR     secs.LowNib
secs10          VAR     secs.HighNib
mins            VAR     Byte                    ' minutes
mins01          VAR     mins.LowNib
mins10          VAR     mins.HighNib
hrs             VAR     Byte                    ' hours
hrs01           VAR     hrs.LowNib
hrs10           VAR     hrs.HighNib
day             VAR     Byte                    ' day

tempIn          VAR     Word                    ' raw temperature
sign            VAR     tempIn.Bit8             ' 1 = negative temperature
tSign           VAR     Bit
```

```
tempC           VAR     Word                    ' Celsius
tempF           VAR     Word                    ' Fahrenheit
tMode           VAR     Bit                     ' 0 = Celsius, 1 = Fahrenheit


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


' -----[ Initialization ]---------------------------------------------------
'
DS1620init:
  HIGH CS_1620                                  ' alert the DS1620
  SHIFTOUT DQ,Clock,LSBFirst,[WrCfg,%10]        ' use with CPU; free-run
  LOW CS_1620
  PAUSE 10
  HIGH CS_1620
  SHIFTOUT DQ,Clock,LSBFirst,[StartC]           ' start conversions
  LOW CS_1620

DS1302init:
  reg = CWPr                                    ' clear write protect register
  ioByte = WPr0
  GOSUB RTC_out

LCDinit:
  PAUSE 500                                     ' let the LCD settle
  DirL = %11111101                              ' setup pins for LCD
  LCDout = %0011                                ' 8-bit mode
  PULSOUT E,1 : PAUSE 5
  PULSOUT E,1
  PULSOUT E,1
  LCDout = %0010                                ' 4-bit mode
  PULSOUT E,1
  ioByte = %00001100                            ' disp on, crsr off, blink off
  GOSUB LCDcommand
  ioByte = %00000110                            ' inc crsr, no disp shift
  GOSUB LCDcommand
  ioByte = ClrLCD
  GOSUB LCDcommand


' -----[ Main Code ]--------------------------------------------------------
'
Main:
  ioByte = 0                                    ' clear las command
  SERIN SIOpin\FCpin, N2400, 200, LCD_Update, [ioByte]
  BRANCH ioByte, [LCD_Update, Reset_Clock, Set_C, Set_F, Send_TmTmp]
  GOTO LCD_Update

Reset_Clock:
```

```
  SERIN SIOpin\FCpin, N2400, 200, LCD_Update, [hrs, mins]
  secs = 0
  GOSUB SetTime                                 ' update DS1302
  GOTO Show_Time                                ' update LCD

Set_C:
  tMode = TM_Cels                               ' celsius
  task = Tsk_Time                               ' do time next time through
  GOTO Show_Temp                                ' update temp display

Set_F:
  tMode = TM_Fahr                               ' fahrenheit
  task = Tsk_Time                               ' do time next time through
  GOTO Show_Temp                                ' update temp display

Send_TmTmp:
  tempIn = tempC                                ' assume in C
  BRANCH tMode, [Send_TmTmp2]                   ' if tMode = 1
  tempIn = tempF                                '   then F

Send_TmTmp2:
  ioByte = "C" + (tMode * 3)                    ' send temp mode indicator
  SEROUT SIOpin\FCpin, N2400, [hrs, mins]
  SEROUT SIOpin\FCpin, N2400, [tempIn.LowByte, tempIn.HighByte, ioByte]
  GOTO Main

LCD_Update:
  BRANCH task, [Show_Time, Show_Temp]

Show_Time:
  GOSUB GetTime
  ioByte = CrsrHm
  GOSUB LCDcommand
  ioByte = hrs10 + "0"  : GOSUB LCDwrite
  ioByte = hrs01 + "0"  : GOSUB LCDwrite
  ioByte = ":"          : GOSUB LCDwrite
  ioByte = mins10 + "0" : GOSUB LCDwrite
  ioByte = mins01 + "0" : GOSUB LCDwrite
  ioByte = ":"          : GOSUB LCDwrite
  ioByte = secs10 + "0" : GOSUB LCDwrite
  ioByte = secs01 + "0" : GOSUB LCDwrite

  task = task + 1 // NumTasks
  GOTO Main

Show_Temp:
  GOSUB GetTemperature                          ' get current temp
  tempIn = tempC                                ' assume C
  BRANCH tMode, [Show_Temp2]                    ' if tMode = 1
  tempIn = tempF                                '   then F
```

```
Show_Temp2:
  ioByte = DDRam + 11                           ' move to right side of LCD
  GOSUB LCDcommand
  ioByte = " "                                  ' space pad (right justify)
  IF (tempIn < 100) THEN Show100
  ioByte = tempIn DIG 2 + "0"

Show100:
  GOSUB LCDwrite
  ioByte = tempIn DIG 1 + "0"
  IF (tempIn > 99) OR (ioByte > "0") THEN Show10
  ioByte = " "

Show10:
  GOSUB LCDwrite

Show01:
  ioByte = tempF DIG 0 + "0" : GOSUB LCDwrite
  ioByte = 223               : GOSUB LCDwrite
  ioByte = "C" + (tMode * 3) : GOSUB LCDwrite

  task = task + 1 // NumTasks
  GOTO Main


' -----[ Subroutines ]-------------------------------------------------------
'
RTC_out:                                        ' send ioByte to reg in DS1302
  HIGH CS_1302
  SHIFTOUT DQ,Clock,LSBFirst,[reg,ioByte]
  LOW CS_1302
  RETURN


RTC_in:                                         ' read ioByte from reg in DS1302
  HIGH CS_1302
  SHIFTOUT DQ,Clock,LSBFirst,[reg]
  SHIFTIN DQ,Clock,LSBPre,[ioByte]
  LOW CS_1302
  RETURN


SetTime:                                        ' write data with burst mode
  HIGH CS_1302
  SHIFTOUT DQ,Clock,LSBFirst,[WrBurst]
  SHIFTOUT DQ,Clock,LSBFirst,[secs,mins,hrs,0,0,day,0,0]
  LOW CS_1302
  RETURN


GetTime:                                        ' read data with burst mode
```

```
  HIGH CS_1302
  SHIFTOUT DQ,Clock,LSBFirst,[RdBurst]
  SHIFTIN DQ,Clock,LSBPre,[secs,mins,hrs,day,day,day]
  LOW CS_1302
  RETURN


GetTemperature:
  HIGH CS_1620                                  ' alert the DS1620
  SHIFTOUT DQ,Clock,LSBFIRST,[RdTmp]            ' give command to read temp
  SHIFTIN DQ,Clock,LSBPRE,[tempIn\9]            ' read it in
  LOW CS_1620                                   ' release the DS1620

  tSign = sign                                  ' save sign bit
  tempIn = tempIn/2                             ' round to whole degrees
  IF tSign = 0 THEN NoNeg1
  tempIn = tempIn | $FF00                       ' extend sign bits for negative

NoNeg1:
  tempC = tempIn                                ' save Celsius value
  tempIn = tempIn */ $01CC                      ' multiply by 1.8
  IF tSign = 0 THEN NoNeg2                      ' if negative, extend sign bits
  tempIn = tempIn | $FF00

NoNeg2:
  tempIn = tempIn + 32                          ' finish C -> F conversion
  tempF = tempIn                                ' save Fahrenheit value
  RETURN


LCDcommand:
  LOW RS                                        ' enter command mode

LCDwrite:
  LCDout = ioByte.HighNib                       ' output high nibble
  PULSOUT E,1                                   ' strobe the Enable line
  LCDout = ioByte.LowNib                        ' output low nibble
  PULSOUT E,1
  HIGH RS                                       ' return to character mode
  RETURN
```