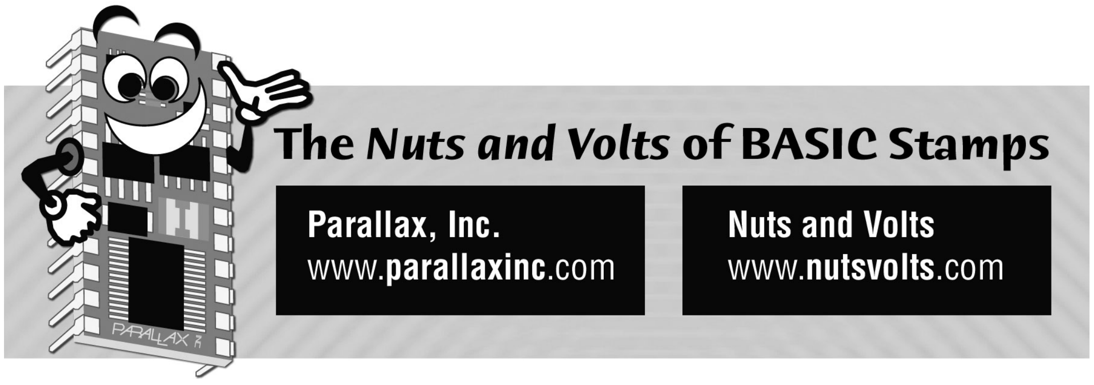


Column #33, November 1997 by Jon Williams:

It’s Time to Get Real Using the Dallas Semiconductor 1302

Have you ever stopped to consider how much of our life is governed by clocks? A bunch. We get up at a given time. We go to work at a given time. We finish work at a given time. Our favorite television and radio programs start and stop at a given time. Wow! Just think how out of sync our society would be without clocks.

A software real-time clock is not a practical exercise when it comes to the BASIC Stamp. Even if one succeeded in implementing the clock function, without interrupts, it's not likely that the Stamp could handle any additional control. It would be too difficult to keep the software clock in sync. It's on occasions like these — when software really isn't the answer — that we turn to external hardware. Thankfully, there are choices. And PBASIC makes them easy to use.

Before we get to hardware specifics, let's talk about time and how to deal with it in a microcontroller. When we refer to time, we're concerned with the hours, minutes, and the appropriate side of noon (AM or PM). This can be a headache to deal with in software. For example, if the time was 11:45 AM and we wanted to add 20 minutes, we'd have to add the 20 to the 45, notice that we went passed 60, carry to the hours variable, check that we went from AM to PM, blah, blah, blah. What a hassle! There is an easier way.

Just a minute, please ….

I once worked for a large irrigation company, and was involved with many teams that designed and built sprinkler timers. Sprinkler timers deal with time on two levels: 1) when to start the sprinklers, and 2) how long the sprinklers should run. It's interesting to note that no matter what the scope of the irrigation controller — from those you can buy at garden centers for residential use, to the big, computerized irrigation systems used on golf courses — time was always dealt with as a single variable. Conversion routines are used to interface with people.

Life for the Stamp would be quite a lot easier if it only had to deal with one variable for the time instead of three. We move from three variables to one by converting the hours, minutes, and AM/PM to minutes. Now you let the Stamp deal with time as a value between zero (12:00 AM) and 1439 (11:59 PM).

It's a reasonably simple matter to convert between this raw time format and something we humans can live with.

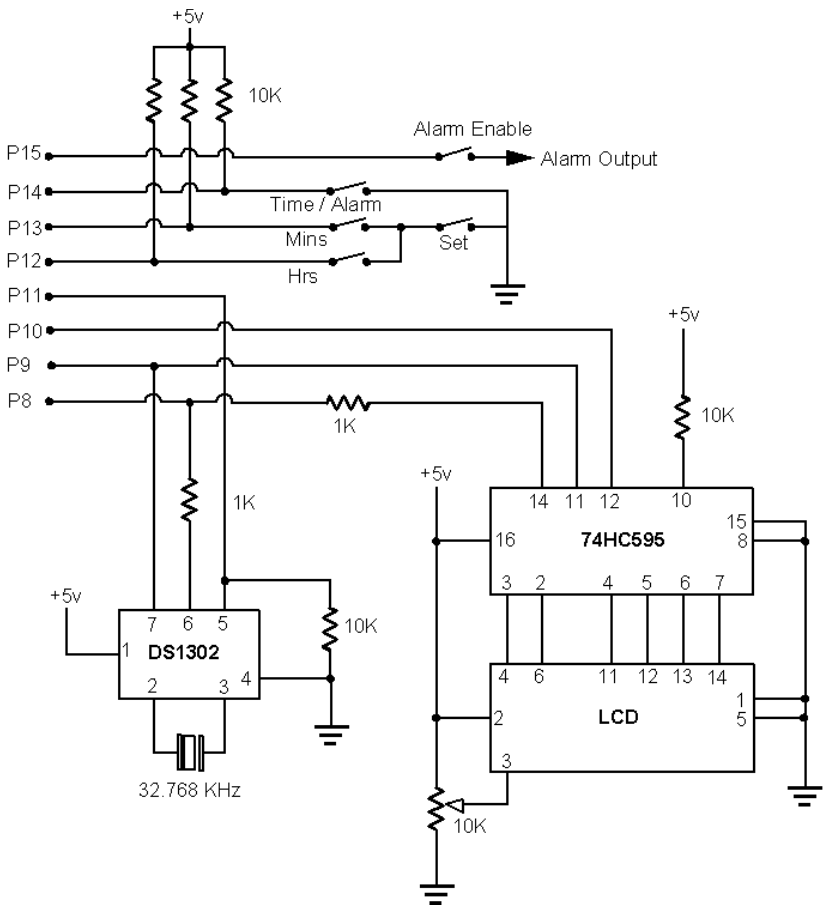
This month, we're going to demonstrate how to deal with real-time by building something that most of us use every day: an alarm clock. We'll use an external RTC chip and take advantage of the LCD interface we covered last month. This project should give you a firm grounding in time-based control, and teach you a math trick or two along the way.

The Dallas Semiconductor DS1302

For our alarm clock, we'll be using the DS1302 from Dallas Semiconductor. This eight-pin DIP keeps track of seconds, minutes, hours (with an AM/PM indicator, if running in 12-hour mode), date of month, month, day of week, and year with leap year compensation valid up to 2100. As a bonus, the DS1302 contains 31 bytes of RAM that we can use as we please. And for projects that use main's power, the DS1302 also contains a trickle-charging circuit that can charge a back-up battery.

Connecting to the Stamp is very straight-forward. The DS1302 uses the same three-wire interface as the DS1620 thermometer (Apr. ’95) and the DS1267 digital pot (Aug. ’96). This allows us to take advantage of the BS2's SHIFTOUT and SHIFTIN commands. To write a value to the DS1302, we'll send (SHIFTOUT) the register address first, then the data for that register. The LSB of the address determines whether the operation is a write (0) or a read (1). To read from the DS1302, we send the address, then immediately read back (SHIFTIN) the register value.

Figure 33.1: DS1302 hookup diagram



For our project, we'll share the clock and data lines with a 74HC595 shift register. This cuts down the number of Stamp pins required to implement the alarm clock. For details on using the 74HC595 with an LCD, please refer to the Oct. '97 issue (More LCDs …).

Time format conversion

In order to keep our interface with the DS1302 simple, we'll use it in the 24-hour mode. In this mode, we don't have to fuss with the AM/PM indicator. (If you want to know how, download DS1302.BS2 from my FTP directory.) For a 12-hour display, we'll deduce AM/PM mathematically.

Another compelling reason to use our raw time format is that the DS1302 stores its registers in BCD (binary coded decimal). In case you haven't dealt with BCD before, here it is in a nutshell: BCD is a method of storing a value between zero and 99 in a byte-sized variable. The ones digit occupies the lower nibble, the tens digit the upper.

Neither nibble of a BCD byte is allowed to have a value greater than nine. Thankfully, the BS2 allows nibble-sized variables and, more importantly, it allows variables to be overlaid. We discussed this technique back in the Aug. '97 issue (PBASIC Programming With Style) and now we're going to use it.

If you study the variables section of Program Listing 33.1, you'll see that we've defined each time element as a byte, then overlaid the tens and ones nibbles. These overlaid nibble definitions do not occupy any more memory in the Stamp. With these definitions, converting the time from the DS1302 to our raw format takes just one line of code:

```
rawTime = (hr10 * 600) + (hr01 * 60) + (mn10 * 10) + mn01
```

Get a pencil and paper, and do it longhand. What you'll find is that the result is always a value between zero and 1439. Keep in mind that we're using 24-hour mode, so the hours value is always between zero and 23.

Converting back is just a bit trickier but, once you get the hang of it, you'll find the technique very useful. The key to this conversion routine is the modulus operator (//). The modulus operator returns the remainder of a division. For example, 5 // 2 (read "five mod two") returns one.

Here's the code to convert from our raw (minutes) format to the BCD values required by the DS1302:

```
hr10 = rawTime / 600
hr01 = rawTime // 600 / 60
mn10 = rawTime // 60 / 10
mn01 = rawTime // 10
```

This first line is easy. Since there are 600 minutes in 10 hours, we simply divide by 600 to get the tens value. Remember that the Stamp uses integer mathematics — the result of a division will always be a whole number. Getting the ones digit of the hours means dividing by 60 since there are 60 minutes in one hour. But first, we've got to get rid of the tens. This is where the modulus operator helps us. Since the Stamp does math from left to right, we first mod the raw value by 600. This step gets rid of the tens of hours. Now we can divide by 60 to get the ones.

Getting the minutes works in the same fashion. First, we mod the raw time by 60 to get rid of all the hours, then we divide by 10 to get the tens digit. The final step is getting to the ones minutes, so we mod the raw time by 10. You'll notice that this technique does not modify the value of the rawTime variable.

Since the nibble variables are overlaid with the time register variables, our BCD bytes are ready to SHIFTOUT to the DS1302. Pretty nifty, isn't it? Unless you've used this technique before, you may not be entirely comfortable with the modulus operator. That's okay, everything gets easier with time (no pun intended). Again, I suggest that you work through the steps with a pencil and paper so that you have a firm grasp of the technique.

Setting the Clock

The modulus operator exhibits a behavior that can be very useful: keeping a variable within a range. The result of x // y will always fall between zero and y-1. And, unlike the MIN and MAX operators, the modulus operator causes the result to wrap around. This table will give you a quick idea of how it works:

```
0 // 3 = 0
1 // 3 = 1
2 // 3 = 2
3 // 3 = 0     - Notice the wrap around
4 // 3 = 1
```

We'll take advantage of this behavior in our time setting routine. Based on the user input, we'll either add one or 60 (an hour) to our raw time and then mod by 1440. This will keep the time between 0 and 1439 and handle the rollover for us. Without the modulus operator, we'd be forced to use a convoluted IF-THEN construct and test for values. No, thank you.

There is one last modulus trick I'd like to share with you before we move on to our clock circuit and software. In the description above, and in the clock we present here, we're simply adding time and rolling over. But what if we wanted to add another input to this project that told the Stamp to subtract time? How would we handle that?

Hang on, this gets a bit gory …. What you'll do to subtract from the raw time is to add the modulus value (1440) minus the amount you want to subtract. So, to subtract a minute, you'll actually add 1439 to the raw time and then apply the modulus of 1440. For example, to subtract an hour from the raw time, you'll add 1380, then apply the modulus of 1440. This technique takes advantage of the wrap-around behavior of the modulus operator. To perform a subtraction, we simply wrap around something less than a full "turn." Let's go through one to see how this works:

- 8:00 PM (20:00) has a raw time value of 1200

- Subtract one minute:
  - 1200 + (1440 - 1) = 2639
  - 2639 // 1440 = 1199
  - {1200 - 1 = 1199}

- Subtract an hour:
  - 1200 + (1440 - 60) = 2580
  - 2580 // 1440 = 1140
  - {1200 - 60 = 1140}

At first, this may look like a lot of work. Please trust me when I tell you that it isn't. These modulus operator tricks will streamline your programs, saving valuable EEPROM space for other code. And it's always a good idea to write efficient code, even when the program is "small." Save yourself a possible rewrite — you never know when a small program will grow into a monster.

Our alarm clock

Now that we've introduced the DS1302 and thoroughly covered the modulus operator, the rest of our alarm clock project is a breeze. My design goal was to use no more than eight of the BS2's I/O pins for the time-keeping and display elements. Four of these pins are used to communicate with the DS1302 and the 74HC595. The rest are used for time-setting buttons and the alarm enable.

To set the clock, press and hold the SET button, then press either the MINUTES or HOURS button to change the time. The BUTTON command debounces the inputs and, with the auto-repeat feature, allows us to change the time quickly by holding either of the time-set buttons. The state of the Time/Alarm input determines which value is displayed and changed. The software is simplified by using an array to hold the raw time values and using the Time/Alarm input as a pointer into the array. When we send the new time to the clock, the seconds value is always reset to zero.

You'll notice that I cheated a bit with the alarm. From a software standpoint, the alarm is always active. This keeps the program simple. The alarm is disabled by opening a switch connected to the alarm pin. We check for an alarm by comparing the current time with the alarm time. This comparison is done inside a loop so that we can activate the alarm pin for any number of minutes we choose (AlrmOut constant) without worrying about crossing midnight.

For alarm outputs shorter than one minute, you can modify the code and use the seconds value returned from the DS1302.

Okay, it's up to you now. Download the DS1302 documentation from Dallas Semiconductor, get out your breadboards, and build the clock. Once you've got it working, you might want to challenge yourself with adding features. How about a second alarm? How about adding everyone's favorite: a snooze button? And why not add another switch to determine the direction (add or subtract) of the hours and minutes time-set buttons? Next month, we'll talk about some possible solutions to these problems.

Time for the BS1?

I can imagine that some BS1 users are feeling just a bit slighted, and I certainly understand. While it is possible to connect the DS1302 to the Stamp 1, the process uses up a lot of code space and doesn't make room for many features. But don't give up, there is another solution. Tune in next month and I'll show you how to build a BS1-based alarm clock with a couple of serial modules. Until then, feel free to use DS1302.BAS from my FTP directory if you'd like to experiment with the DS1302.

```
' Program Listing 33.1
' Stamp Applications: Nuts & Volts, November 1997

' ----[ Title ]-----------------------------------------------------------
'
' File...... BS2CLOCK.BS2
' Purpose... Stamp II-based Alarm Clock
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' WWW....... http://members.aol.com/jonwms
' Started... 20 SEP 97
' Updated... 27 SEP 97


' ----[ Program Description ]---------------------------------------------
'
' This program demonstrates basic timekeeping functions using the DS1302
' from Dallas Semiconductor. In order to minimize the number of I/O pins
' used, a 74HC595 is used to send data to an LCD. The DS1302 and 74HC595
' share the Dio and Clock lines.


' ----[ Revision History ]------------------------------------------------
'
' 26 SEP 97 : Timekeeping functions complete
' 27 SEP 97 : Alarm function complete


' ----[ Constants ]-------------------------------------------------------
'
' ===================
' I/O Pin Definitions
' ===================
'
Dio     CON        8                ' DS1302.6, 74HC595.14
Clk     CON        9                ' DS1302.7, 74HC595.11
CS_595  CON       10                ' 74HC595.12
CS_1302 CON       11                ' DS1302.5
SetHr   CON       12
SetMn   CON       13
TmAlrm  VAR      In14
Alarm   VAR     Out15


' ================
' DS1302 Registers
' ================
'
WrSc    CON    $80                  ' write seconds
RdSc    CON    $81                  ' read seconds
```

```
WrMn    CON     $82
RdMn    CON     $83
WrHr    CON     $84
RdHr    CON     $85

CWPr    CON     $8E                     ' write protect register
WPr1    CON     $80                     ' set write protect
WPr0    CON     $00                     ' clear write protect

WrBrst  CON     $BE                     ' write burst of data
RdBrst  CON     $BF                     ' read burst of data

WrRam   CON     $C0                     ' RAM address control
RdRam   CON     $C1

T24hr   CON     0                       ' 24 hour clock mode
T12hr   CON     1                       ' 12 hour clock mode
AM      CON     0
PM      CON     1

AlrmLen CON     1                       ' length of alarm (in minutes)


' ======================
' LCD control characters
' ======================
'
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
CGRam   CON     $40                     ' Char Gen RAM control


' ----[ Variables ]-------------------------------------------------------
'
addr    VAR     Byte                    ' DS1302 address to read/write
ioByte  VAR     Byte

secs    VAR     Byte
sc10    VAR     secs.HIGHNIB
sc01    VAR     secs.LOWNIB
mins    VAR     Byte
mn10    VAR     mins.HIGHNIB
mn01    VAR     mins.LOWNIB
hrs     VAR     Byte
hr10    VAR     hrs.HIGHNIB
hr01    VAR     hrs.LOWNIB
```

```
ampm    VAR     hrs.Bit5                ' 0 = AM, 1 = PM
tMode   VAR     hrs.Bit7                ' 0 = 24, 1 = 12

rawTime VAR     Word(2)                 ' raw storage of time values
work    VAR     Word                    ' work variable for display output
oldSc   VAR     Byte                    ' previous seconds value
apChar  VAR     Byte                    ' "A" (65) or "P" (80)

char    VAR     Byte                    ' character to send to LCD
temp    VAR     Byte                    ' work variable for LCD routine
lcd_E   VAR     temp.Bit2               ' LCD Enable pin
lcd_RS  VAR      temp.Bit3              ' Register Select (1 = char)

butn    VAR     Byte                    ' BUTTON workspace variable
alrmX   VAR     Bit


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


' ----[ Initialization ]-------------------------------------------------
'
Init:   DirH = %10001110

        addr = CWPr                     ' clear write protect register
        ioByte = WPr0
        GOSUB RTCout

        oldSc = $99                     ' set the display flag
        tMode = T24Hr                   ' put clock in 24-hour mode
        rawTime(0) = 360                ' preset alarm to 6:00 AM
        GOSUB SetRTm                    ' set time to 12:00 AM

' Initialize the LCD (Hitachi HD44780 controller)
'
LCDini: PAUSE 500                       ' let the LCD settle
        char = %0011                    ' 8-bit mode
        GOSUB LCDcmd
        PAUSE 5
        GOSUB LCDcmd
        GOSUB LCDcmd
        char = %0010                    ' put in 4-bit mode
        GOSUB LCDcmd
        char = %00001100                ' disp on, crsr off, blink off
        GOSUB LCDcmd
        char = %00000110                ' inc crsr, no disp shift
        GOSUB LCDcmd
        char = ClrLCD
        GOSUB LCDcmd
```

```
' ----[ Main Code ]---------------------------------------------------
'
Start:  GOSUB GetTm
        IF secs = oldSc THEN ChkHr
        GOSUB ShowTm

ChkHr:  BUTTON SetHr,0,150,10,butn,0,ChkMn   ' is Set Hours pressed?
        GOSUB GetTm                          ' yes, get the clock
        rawTime(TmAlrm) = rawTime(TmAlrm)+60//1440
        IF TmAlrm = 0 THEN NoSet1            ' skip 1302 set for alarm
        GOSUB SetRTm                         ' set new time
NoSet1: GOSUB ShowTm                         ' display the change
        PAUSE 100                            ' pause between changes
        GOTO ChkHr                           ' still pressed?

ChkMn:  BUTTON SetMn,0,150,10,butn,0,ChAlrm  ' is Set Mins pressed?
        GOSUB GetTm
        rawTime(TmAlrm) = rawTime(TmAlrm)+1//1440
        IF TmAlrm = 0 THEN NoSet2
        GOSUB SetRTm
NoSet2: GOSUB ShowTm
        PAUSE 100
        GOTO ChkMn

ChAlrm: IF AlrmLen = 0 THEN Start            ' skip if no alarm length
        alrmX = 0                            ' assume no alarm
        FOR temp = 0 TO (AlrmLen-1)          ' check for length of alarm
          work = rawTime(0)+temp//1440       ' calculate alarm range
          IF rawTime(1) <> work THEN NoAlrm  ' is time in range?
          alrmX = 1                          ' yes
NoAlrm: NEXT
        Alarm = alrmX                        ' output the alarm state

        GOTO Start                           ' start all over


' ----[ Subroutines ]---------------------------------------------------
'
' send a byte (ioByte) to the DS1302 location specified by addr
'
RTCout: HIGH CS_1302


' get a byte (ioByte) from the DS1302 location specified by addr
'
RTCin:  HIGH CS_1302
        SHIFTOUT Dio,Clk,LSBFIRST,[addr]
        SHIFTIN Dio,Clk,LSBPRE,[ioByte]
        LOW CS_1302
        RETURN

' convert raw time format to BCD bytes for DS1302
'
SetRTm: hr10 = rawTime(1) / 600
        hr01 = rawTime(1) // 600 / 60
        mn10 = rawTime(1) // 60 / 10
        mn01 = rawTime(1) // 10
        secs = $00

' use burst mode to set the clock and calendar
' -- do not remove the third SHIFTOUT line
' -- you must send 8 bytes for data to be written in burst mode
'
SetTm:  HIGH CS_1302
        addr = WrBrst
        temp = 0
        SHIFTOUT Dio,Clk,LSBFIRST,[addr]
        SHIFTOUT Dio,Clk,LSBFIRST,[secs,mins,hrs]
        SHIFTOUT Dio,Clk,LSBFIRST,[temp,temp,temp,temp,temp]
        LOW CS_1302
        RETURN

' use burst mode to the grab time (hrs, mins & secs)
'
GetTm:  HIGH CS_1302
        addr = RdBrst
        SHIFTOUT Dio,Clk,LSBFIRST,[addr]
        SHIFTIN Dio,Clk,LSBPRE,[secs,mins,hrs]
        LOW CS_1302
        rawTime(1) = ((hr10 & %11)*600)+(hr01*60)+(mn10*10)+mn01
        RETURN

' Send command to the LCD
'
LCDcmd: lcd_RS = 0                    ' command mode
        GOTO LCDout

' Write ASCII char to LCD
'
LCDwr:  lcd_RS = 1                    ' character mode


LCDout: temp.HIGHNIB = char.HIGHNIB   ' get high nibble
        lcd_E = 1
        SHIFTOUT Dio, Clk, MSBFIRST, [temp]
        PULSOUT CS_595, 1
        lcd_E = 0                     ' drop Enable line low
        SHIFTOUT Dio, Clk, MSBFIRST, [temp]
        PULSOUT CS_595, 1
        temp.HIGHNIB = char.LOWNIB    ' get low nibble
        lcd_E = 1
        SHIFTOUT Dio, Clk, MSBFIRST, [temp]
        PULSOUT CS_595, 1
        lcd_E = 0
        SHIFTOUT Dio, Clk, MSBFIRST, [temp]
        PULSOUT CS_595, 1
        RETURN


' Show time in LCD
' -- the time displayed is controlled by the position
' -- of the TmAlrm switch input
'
ShowTm: char = DDRam+$02              ' move to third position in LCD
        GOSUB LCDcmd

        work = rawTime(TmAlrm)        ' get the raw time
        hrs = work / 60               ' extract (decimal) hours
        mins = work // 60             ' extract (decimal) minutes
        IF TmAlrm = 1 THEN ST0
        secs = 0                      ' show zero seconds if alarm

ST0:    apChar = "A"
        IF hrs > 0 THEN ST1
        hrs = 12                      ' zero hours --> 12 AM
        GOTO ST3

ST1:    IF hrs > 11 THEN ST2
        GOTO ST3

ST2:    apChar = "P"
        IF hrs = 12 THEN ST3
        hrs = hrs-12                  ' 13 - 23 --> 1 - 11 PM

ST3:    char = hrs DIG 1+48           ' get hours, convert to ASCII
        IF char <> "0" THEN ST4
        char = " "                    ' remove leading zero

ST4:    GOSUB LCDwr                   ' write hours.tens
        char = hrs DIG 0+48
        GOSUB LCDwr                   ' write hours.ones
        char = ":"
        GOSUB LCDwr
        char = mins DIG 1+48
        GOSUB LCDwr
        char = mins DIG 0+48
        GOSUB LCDwr
        char = ":"
        GOSUB LCDwr
        char = sc10+48
        GOSUB LCDwr
```

```
       char = sc01+48
       GOSUB LCDwr
       char = " "
       GOSUB LCDwr
       char = apChar
       GOSUB LCDwr
       char = "M"
       GOSUB LCDwr

       oldSc = secs                  ' reset check value
EndST:  RETURN
```