



Column #34, December 1997 by Jon Williams:

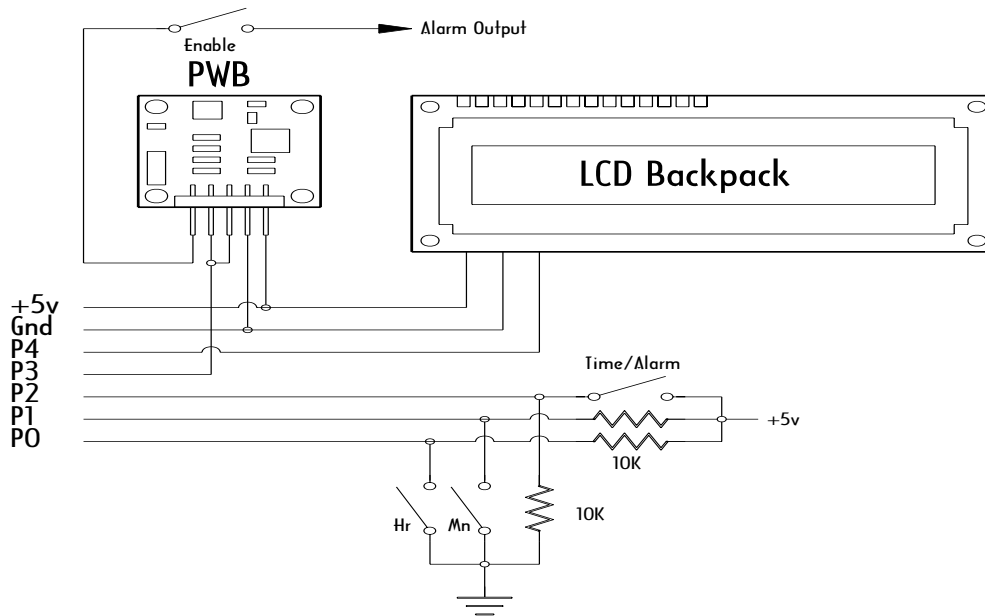
## It's Time to Get Real Using the Dallas Semiconductor 1302 - Part 2

Last month, we built a working alarm clock with the Stamp 2 and the Dallas Semiconductor DS1302. While I've been able to port the heart of that code to the Stamp 1, there just isn't enough program space left to do anything practical. This month, we'll create an alarm clock with the Stamp 1 by using a recent addition to the list of serial saviors: the Pocket Watch B (PWB) from Solutions Cubed. And, of course, we'll call on our old friend — the Scott Edwards' serial LCD backpack — to provide the display for the clock.

The Pocket Watch B is a pin-for-pin replacement of the original Pocket Watch that Scott reviewed in this column in April '97. Improvements include:

- Factory calibrated timebase
- User accessible timebase
- User calibration of timebase
- Three advanced alarms: single-shot, short astable, and long astable
- Year 2000 compliance
- PCB insertable SIP connector
- Streamlined communications protocol

Figure 34.1: Pocket Watch B hookup diagram



Physically, the Pocket Watch B is a one-inch-squared circuit board with a five-pin SIP connector on one edge. I found the arrangement of the SIP connector very handy; I simply plugged the PWB into a solderless breadboard and started experimenting. In a permanent project, you could use Scott Edwards' connectamundos header-post jumper wires (I love those things!). Refer to the Oct. '95 installment of Stamp Applications for instructions on building your own connectamundos-type jumper wires.

The connections to the PWB are +5 volts, ground, transmit, receive, and alarm. Don't be worried about the separate transmit and receive pins. These pins are never active at the same time, so we can tie them together for use with the Stamp. The alarm output from the PWB is a real help in conserving our I/O resources. Last month, we used eight pins. This month, we only need five.

### How It Works

The PWB is a stand-alone, time-keeping module with advanced alarm features. It contains separate registers for the time and alarm values. Last month, we were forced to

synthesize an alarm with the DS1302. The PWB conveniently handles all those details for us. The PWB also differs from the DS1302 in that the registers are transmitted in straight binary. We don't need to worry about BCD conversions when using the PWB.

Like the LCD backpack, the PWB communicates with the Stamp through an asynchronous serial connection. When you examine the code, you'll notice that each command sequence begins with \$55. This byte is necessary for the PWB to detect the baud rate being used. And, while 4800 and 9600 baud are supported, the standard speed of the Stamp 1 limits us to 2400 baud.

The PWB uses two different time modes: standard and extended. In extended mode, two bytes are used for the year. The high byte of the year contains the century value (i.e., 19). If you need year-2000 compliance, or want to use one of the advanced alarms, use extended mode. And keep in mind that the PWB operates strictly in 24-hour mode. That is, the hours will always be from zero to 23. In order to save space, we'll stick with that format for this project. If you'd like to have a 12-hour clock, refer to last month's code for the conversion routine.

In addition to the physical alarm output, the PWB has four distinct alarm types: standard, single-short, short astable, and long astable. The standard alarm is the easiest to understand, and yet, the least convenient to use. Once the standard alarm trips, it will not reset unless the PWB power is cycled or an "alarm off" command is sent to the PWB. Either case requires the Stamp to monitor the alarm output, using up another precious pin.

The single-shot alarm output causes the alarm output to be on for a user-specified number of seconds. The PWB accepts values \$0000 to \$FFFE (65,534) allowing for very long outputs (greater than 18 hours). A possible down side to the single-shot mode is that it is not automatically re-enabled once fired. This means that you have to reset the alarm after the single-shot output has activated. One last note: The PWB automatically adds one second to the time you download. Keep this in mind for precision applications.

As I pointed out earlier, the PWB has two astable alarms: short and long. They are similar in that you download a repetition rate and duration for the alarm output. In short mode (\$17), repetition is specified in minutes, and duration is specified in seconds. In long mode (\$18), repetition is specified in hours, and output is in minutes. And just as with the single-shot mode, the PWB increments each register (repetition and time) by one. Since our project is a standard alarm clock, we tell the PWB to turn on the alarm output for one minute every 24 hours with this command:

## Column #34: It's Time to Get Real Using the Dallas Semiconductor 1302 - Part 2

SEROUT ClkIO, ClkBaud, (\$55, \$18, 23, 0)

### Our BS1 Alarm Clock

Aside from the different RTC and LCD hardware, our BS1 alarm clock is structurally similar to the BS2 version we built last month. The features of the PWB certainly simplify the code, allowing it to fit into the slim resources of the Stamp 1.

Examining Program Listing 34.1, you'll see that the code reads the PWB, displays the time just read, and checks the time-set buttons. If either time-set button was pressed, the raw time value is updated and the clock is reset with that value. Since the PWB handles the alarm processing, we simply need to connect our circuit to it.

There are no new tricks used in this program, however, there is one area that I feel could use a bit more explanation: the use of I/O pins as variables. If you look very carefully, you'll see that this program does not actually know what it's reading or setting (time or alarm). The program simply deals with a time value. What this value represents is determined by the state of the Time/Alarm input.

Within our definitions, we have a variable called TmAlrm that tracks the state of pin 2. This pin is normally pulled low through a 10K resistor. With the Time/Alarm switch in the Time position (open), a read of TmAlrm will return a value of zero. When the switch is in the Alarm position (closed), a read of TmAlrm will return a value of one. The value of TmAlrm is refreshed every time we use it since it is referencing pin 2.

Last month, we used this input (albeit inverted) as an array index for our rawTime variable. Since the PWB contains separate time and alarm registers, this month we're using TmAlrm variable to determine which set of registers are being retrieved or set. To read the time registers (standard mode is suitable for our project) from the PWB, we need to issue the command byte of \$02. To read the alarm time registers, we need to issue a command byte of \$03. No problem; we simply add the value of TmAlrm to \$02 and use this as our command byte.

We use the same technique when setting the time or alarm. There is a bit of extra code in the SetTm routine to set up the long astable alarm and to enable the alarm output. The PWB command set made the technique I just described pretty easy. But what would you do if the commands were, say, \$03 and \$15? This is still not a problem; it just requires a little extra math. If the code for "get time" was \$03, and the code for "get alarm" was \$15, we'd rewrite our command line like this:

### Column #34: It's Time to Get Real Using the Dallas Semiconductor 1302 - Part 2

```
temp1 = $12 * TmAlarm + $03
```

Run through the math so you understand how this works. Note that the order is important. Remember that the Stamp 1 evaluates math strictly from left to right — parenthesis are not allowed and there are no operator precedents. Do keep this one in your bag of tricks. It'll save you a couple of lines of code, which is always important when working with the Stamp 1.

True to our word, we built an alarm clock with the Stamp 1. It's not particularly fancy, but it is very functional and there's enough space left for a few bells and whistles. If you connect the PWB to a Stamp 2, you'll have plenty of code space to work with the date registers. Give it a try.

Once again, I'd like to thank those of you who have sent E-Mail. I sincerely appreciate your kind comments and your thoughtful ideas. And I'd like to thank the good folks at Nuts & Volts and my friend Scott Edwards for their faith in my first stint as a columnist. To each of you reading this, may God bless you, your families, and loved ones through this holiday season, and the coming new year. I'll see you in 1998!

## Column #34: It's Time to Get Real Using the Dallas Semiconductor 1302 - Part 2

```
' Program Listing 34.1
' Stamp Applications: Nuts & Volts, December 1997

' ----[ Title ]-----
'
' File..... BS1CLOCK.BAS
' Purpose... Stamp 1-based Alarm Clock
' Author.... Jon Williams
' E-mail.... jonwms@aol.com
' WWW..... http://members.aol.com/jonwms
' Started... 25 OCT 97
' Updated... 26 OCT 97

' ----[ Program Description ]-----
'
' This program demonstrates basic timekeeping functions using the Pocket
' Watch B real-time clock module from Solutions Cubed. A Scott Edwards'
' Electronics serial backpack is used for the display

' ----[ Revision History ]-----
'
' 26 OCT 97 : Completed and working

' ----[ Constants ]-----
'
' =====
' I/O Pin Definitions
' =====
'
SYMBOL SetHr   = 0
SYMBOL SetMn   = 1
SYMBOL TmAlrm = Pin2

' =====
' Pocket Watch B
' =====
'
SYMBOL ClkIO   = 3
SYMBOL ClkBaud = T2400

' command set
SYMBOL WrTime = $00      ' write time -- standard
SYMBOL WrAlrm = $01      ' write alarm -- standard
SYMBOL RdTime = $02      ' read time -- standard
SYMBOL RdAlrm = $03      ' read alarm -- standard
SYMBOL Alrm1  = $04      ' alarm on
SYMBOL Alrm0  = $05      ' alarm off
```

## Column #34: It's Time to Get Real Using the Dallas Semiconductor 1302 - Part 2

```

SYMBOL WrXTm = $10      ' write time -- extended
SYMBOL WrXAlrm= $11    ' write alarm -- extended
SYMBOL RdXTm = $12     ' read time -- extended
SYMBOL RdXAlrm= $13   ' read alarm -- extended
SYMBOL Alrm1X = $14    ' alarm on -- extended
SYMBOL Alrm0X = $15    ' alarm off -- extended
SYMBOL SSAAlrm = $16   ' single-shot alarm
SYMBOL SAAAlrm = $17   ' short astable alarm
SYMBOL LAAAlrm = $18   ' long astable alarm
SYMBOL GetAlrm= $19   ' get alarm characteristics

' =====
' SEE Serial Backpack LCD
' =====
'
SYMBOL LcdIO = 4
SYMBOL LcdBaud= N2400

SYMBOL I = $FE        ' backpack instruction toggle
SYMBOL ClrLCD = $01   ' clear the LCD
SYMBOL CrsrHm = $02   ' move cursor to home position
SYMBOL CrsrLf = $10   ' move cursor left
SYMBOL CrsrRt = $14   ' move cursor right
SYMBOL DispLf = $18   ' shift displayed chars left
SYMBOL DispRt = $1C   ' shift displayed chars right
SYMBOL DDRam = $80    ' Display Data Ram control
SYMBOL CGRam = $40    ' Char Gen Ram control

SYMBOL BtnDly = 35    ' delay for BUTTON loop

' ---- [ Variables ]-----
'
SYMBOL secs = B0
SYMBOL mins = B1
SYMBOL hrs = B2
SYMBOL xx = B3        ' space holder for date registers
SYMBOL temp1 = B4
SYMBOL temp2 = B5
SYMBOL butn = B6      ' BUTTON workspace
SYMBOL rawTm = W4     ' raw time (minutes past midnight)

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

' ---- [ Initialization ]-----
'
Init: Dirs = %00011000

```

## Column #34: It's Time to Get Real Using the Dallas Semiconductor 1302 - Part 2

```

Pins = %00001000
PAUSE 1000                                ' let clock and LCD initialize

' ----[ Main Code ]-----
'
Start: GOSUB GetTm                          ' get the clock
      GOSUB ShowTm                          ' show clock on backpack lcd

      butn = 0

ChkHr: BUTTON SetHr,0,150,10,butn,0,ChkMn  ' is Set Hours pressed?
      GOSUB GetTm                            ' yes, get the clock
      rawTm = rawTm + 60 // 1440
      GOSUB SetTm                            ' set new time
      GOSUB ShowTm                          ' display the change
      PAUSE BtnDly                          ' pause between changes
      GOTO ChkHr                            ' still pressed?

ChkMn: BUTTON SetMn,0,150,10,butn,0,Start  ' is Set Minutes pressed?
      GOSUB GetTm                            ' yes, get the clock
      rawTm = rawTm + 1 // 1440
      GOSUB SetTm                            ' set new time
      GOSUB ShowTm                          ' display the change
      PAUSE BtnDly                          ' pause between changes
      GOTO ChkMn                            ' still pressed?

      GOTO Start

' ----[ Subroutines ]-----
'
GetTm: temp1 = $02 + TmAlrm                  ' time ($02) or alarm ($03)
      SEROUT ClkIO,ClkBaud,($55,temp1)
      ' note that we ignore day, month and year
      SERIN  ClkIO,ClkBaud,secs,mins,hrs,xx,xx,xx
      rawTm = hrs * 60 + mins                ' update raw time value
      RETURN

SetTm: hrs = rawTm / 60                     ' get hours from raw time
      mins = rawTm // 60                    ' get minutes from raw time
      ' set the clock time
      ' (alarm if "TmAlrm" input is high)
      temp1 = $10 + TmAlrm                  ' calculate command code
      SEROUT ClkIO,ClkBaud,($55,temp1,0,mins,hrs,10,18,97)
      IF TmAlrm = 0 THEN Done
      ' reset the alarm output duration
      ' every 24 hours for 1 minute
      ' (the Pocket Watch B increments each value by 1)
      SEROUT ClkIO,ClkBaud,($55,LAAlrm,23,0)
      ' enable the alarm
      SEROUT ClkIO,ClkBaud,($55,Alrm1X)

```

## Column #34: It's Time to Get Real Using the Dallas Semiconductor 1302 - Part 2

```
Done:  RETURN

ShowTm: SEROUT LcdIO,LCDBaud,(I,ClrLCD)      ' clear the lcd
        temp1 = hrs / 10                    ' get the hours 10s digit
        temp2 = hrs // 10                   ' get the hours 1s digit
        SEROUT LcdIO,LCDBaud, (#temp1,#temp2,":")
        temp1 = mins / 10                   ' get the mins 10s digit
        temp2 = mins // 10                  ' get the mins 1s digit
        SEROUT LcdIO,LCDBaud, (#temp1,#temp2)
        ' remove next 3 lines to conserve code space
        temp1 = secs / 10                   ' get the secs 10s digit
        temp2 = secs // 10                  ' get the secs 1s digit
        SEROUT LcdIO,LCDBaud, (":",#temp1,#temp2)
        RETURN
```

