AP-02, an accurate timebase

April 2003, cws/alm

This paper contains three programs:

1) An appnote for the BASIC Stamp I.
   Look for the original at www.parallax.com.

2) AmrBASIC adaption of the same program.
   See the current executable version in
   ~/amrforth/v6/017/basic for Linux, or
   \amrforth\v6\017\basic for Windows.

3) AmrForth adaption of the same program.
   See the current executable version in
   ~/amrforth/v6/017/example for Linux, or
   \amrforth\v6\017\example for Windows.

Programs are listed at the end of the paper, with line numbers for reference.

Let's look at the original appnote program for the BASIC Stamp I. A counter/timer with a 32 KHz crystal is attached to pin0, so that pin0 toggles four times per second. Each time pin0 toggles it is detected at line 9, when pin0 is different from bit0. The program then jumps to the tick label at line 22. B0 is incremented in line 23, which makes bit0 equal to pin0 by the way. On line 24, if the fourth toggle has not yet occurred, then we jump back to Main. If the fourth toggle has occurred, then we know that one second has passed. We set b0 back to 0 in line 25. Line 26 increments the seconds counter, in w1. W1 is used instead of w0 because bit0 is overlayed on w0. Line 27 uses the debug command to display the number of seconds elapsed over a serial port. Line 28 jumps back to Main. Lines 10-12 of Main contain comments suggesting that other program code could go there, and be executed concurrently with the counting of seconds.

We decided to use an internal counter to keep time in our amrBASIC program. The Cygnal f017 board uses a 24 MHz crystal, which gives us a very accurate millisecond timer for the BASIC PAUSE command. We added server BASIC commands to allow access to the millisecond timer as a free running timer. This eliminates any lost time during the reading or updating of the timer. In other words, instead of waiting for the timer to count down to zero, then reloading the timer, we let the timer count down to 1000, then add another 1000 to it to keep it going. It never reaches zero, never stops, and doesn't accumulate any error due to the time it takes to reload the timer.

Program #2 shows how to use the new BASIC commands. Line 5 starts with the label 'entry'. This is the entry point of the program. On line 16 you will see 'RUN entry' which installs 'entry' as the entry point. In other words, the 'entry' program becomes the default program after reset.

The line containing 'entry' uses the command 'timing 2000'. The 'timing' command is special to amrBASIC. It enables the millisecond timer interrupt and loads the millisecond counter with 2000.

Left alone the millisecond interrupt would count down to zero then disable itself. This is how 'pause' works. When we use 'timing' instead of 'pause', we get back control of the device immediately and we can poll the timer to see how much time has elapsed. The amrBASIC has a special word variable called 'timer' which can be used to read the millisecond timer.

Remember that the timer counts down.

```
\ Program #1
\ --- The original demo program.
     1  ' Program: TIC_TOC.BAS (Increment a counter in response to a
     2  ' precision 2-Hz external clock.
     3
     4  ' The 2-Hz input is connected to pin0.  Bit0 is the lowest bit of b0,
     5  ' so each time b0 is incremented (in tick), bit0 gets toggled.  This
     6  ' ensures that tick gets executed only once per transition of pin0.
     7
     8  Main:
     9    if pin0 = bit0 then tick
    10    ' Other program activities--
    11    ' up to 250 ms worth --
    12    ' go here.
    13    goto Main
    14
    15 ' Tick maintains a 16-bit counter to accumulate the number of seconds.
    16 ' The maximum time interval w1 can hold is 65535 seconds--a bit over
    17 ' 18 hours.  If you want a minute count instead, change the second
    18 ' line of tick to read: "if b0 < 240 then Main".  There are 1440 minutes
    19 ' in a day, so w1 can hold up to 65535/1440 = 45.5 days worth of to-the-
    20 ' minute timing information.
    21
    22 tick:
    23    let b0 = b0 + 1              ' Increment b0 counter.
    24    if b0 < 4 then Main          ' if b0 hasn't reached 4, back to Main.
    25    let b0 = 0                   ' Else clear b0,
    26    let w1 = w1 + 1              ' increment the seconds count,
    27    debug cls,#w1," sec."        ' and display the seconds.
    28    goto Main           ' Do it again.

\ Program #2
\ Can be found at ~/amrforth/v6/017/basic/timing.bas
\ or              \amrforth\v6\017\basic\timing.bas
\ in its current executable form.
\ --- Our BASIC verision.
     1    BASIC
     2    ' timing.bas  An example application using a very accurate free running
     3    ' timer.
     4
     5    entry:    timing 2000        ' Start timer and set to 2000 ms.
     6              let w0 = 0         ' Initialize seconds counter.
     7    Main:     if timer <= 1000 then tick   ' One second has passed.
     8              ' Do something that takes less than 1 second, if you like.
     9              goto Main
    10
    11    tick:     add-timer 1000     ' Add one second to the timer.
    12              let w0 = w0 + 1              ' Add one to the seconds count.
    13              print w0           ' Show the second count.
    14              goto Main
    15
    16    RUN entry
    17
    18    END
```

```
\ Program #3
\ Can be found at ~/amrforth/v6/017/example/timing.fs
\ or              \amrforth\v6\017\example\timing.fs
\ in its current executable form.
\ --- Our Forth version.
     1    \ timing.fs
     2    \ An example application using a very accurate free running timer.
     3
     4    include basic.fs  \ For the ms timer interrupt.
     5    \ See \amrforth\v6\basic.fs for definitions of:
     6    \ ms-counter       A variable.
     7    \ atomic-@         Disables interrupts while fetching.
     8    \ add-timer        Adds to the ms-counter while interrupts are disabled.
     9    \ set-mstimer      Starts the millisecond timer.
    10
    11    in-meta  \ Forth metacompiler vocabulary.
    12
    13    \ Fetch the milliseconds counter with interrupts disabled.
    14    : timer  (  - n) ms-counter atomic-@ ;
    15
    16    \ Increment the seconds counter and display it.
    17    : tick  ( n1 - n2) 1000 add-timer  1 +  dup . ;
    18
    19    : main  (  - )
    20             2000 set-mstimer  \ Start the millisecond timer.
    21             0  \ Seconds count rides on the data stack.
    22             begin    timer 1001 < if  tick  then  \ One second has passed.
    23                      \ Do something here that doesn't take too long,
    24                      \ it may delay the execution of tick.  Since the timer
    25                      \ is free running, the delay won't affect the overall
    26                      \ accuracy of the timer, just the individual display.
    27             again ;
    28
```