A Virtual Peripheral Time Clock

Introduction

This application note presents programming techniques for implementing a real time clock that keeps a 16-bit milliseconds count, and has the option for full time clock capabilities, including seconds, minutes, hours, and days. The routine takes advantage of the SX's internal interrupt feature to allow background operation of the clock as a virtual peripheral.

How the code works

This firmware module requires no external circuitry (other than an oscillator crystal) and is quite straight forward. There are three options for code assembly, controlled by the *clock_type* parameter. If *clock_type=0*, the clock only counts milliseconds. A value of *clock_type*=1 adds seconds, minutes and hours, and a value of *clock_type*=2 allows the days to be counted as well.

The timing constant for each millisecond 'tick' is determined as follows:

msec tick timing = osc. freq. / (1000 msec/sec * prescaler * mode) where mode=1 (turbo) or =4 (normal)

So, for a crystal frequency of 50 MHz, in turbo mode, with a prescaler of 1, the msec tick timing constant is:

msec tick timing = $50 \times 10^6 / (1000 * 1 * 1) = 50 \times 10^3$

By comparing the number of elapsed instructions with the millisecond tick timing, the code decides when one millisecond has passed and increments the *msc_lo* and *msec_hi* counters accordingly. In the same way, if selected, the code checks the corresponding count and ticks off each second, minute, hour, day, etc.

The time clock's accuracy is dependent upon the accuracy of the oscillator used, which for crystals is usually extremely good. For oscillators, especially slower ones, that do not have a frequency in kHz that is divisible by an integer, the accuracy starts being affected by the msec count timing algorithm, and it should be adjusted or left out accordingly[1].

Modifications and further options

Ideally the circuit and program will provide a method for the user to enter the time and date, etc., otherwise it is a relative time count in reference to the last time the circuit was turned on or reset.

If the need for processor power between timed events is minimal, the routine could be modified and set up in conjunction with the watchdog timer instead of the internal RTCC interrupt where the SX is put in sleep mode between watchdog time-outs. This allows for a tremendous savings in power consumption.

With some additional programming, day-of-the-week, month, and even year counts could be added, the month count being somewhat more involved, for obvious reasons.

---

[1] With a 32768Hz watch crystal, for example, this is not the optimal algorithm. The msec count could be dropped and only the seconds (and larger) counts kept to resolve this issue.

Program Listing

```
;*****************************************************************************
;       Software Time Clock
;
;
;       Length: 28/50/56 bytes (depending upon clock type, +1 for bank select)
;       Author: Craig Webb
;       Written: 98/8/17
;
;       This program implements a software time clock virtual peripheral
;       that keeps a 16 bit count of elapsed time in milliseconds.
;       The option is available to include seconds, minutes, hours and even
;       days to this clock if desired.
;       The code takes advantage of the SX's internal RTCC-driven interrupt
;       to operate in the background while the main program loop is executing.
;
;*****************************************************************************
;
;****** Assembler directives
;
; uses: SX28AC, 2 pages of program memory, 8 banks of RAM, high speed osc.
;       operating in turbo mode, with 8-level stack & extended option reg.
;
              DEVICE  pins28,pages2,banks8,oschs
              DEVICE  turbo,stackx,optionx
              ID      'TimeClck'                  ;program ID label
              RESET   reset_entry                 ;set reset/boot address
;
;****************************** Program Variables *************************
;
;****** Program Parameters
;
;clock_type    =       0                   ;16 bit msec count only
clock_type     =       1                   ;include sec, min, hours
;clock_type    =       2                   ;include day counter
;
;****** Program Constants
;
tick_lo        =       80                  ;50000 = msec instruction count
tick_hi        =       195                 ; for 50MHz, turbo, prescaler=1
;
int_period     =       163                 ;period between interrupts
;
mspersec_hi    =       1000/256            ;msec per second hi count
mspersec_lo    =       1000-(mspersec_hi*256) ;msec per second lo count
;
;****** Register definitions
;
              org     8                   ;start of program registers
main           =      $                   ;main bank
;
temp           ds     1                   ;temporary storage
;
              org     010H                ;bank0 variables
clock         EQU     $                   ;clock bank
;
time_base_lo   DS      1                   ;time base delay (low byte)
time_base_hi   DS      1                   ;time base delay (high byte)
msec_lo        DS      1                   ;millisecond count (low)
msec_hi        DS      1                   ;millisecond count (high)

              IF      clock_type>0        ;do we want sec, min, hours?
seconds        DS      1                   ;seconds count
minutes        DS      1                   ;minutes count
hours          DS      1                   ;hours count
```

```
                ENDIF

                IF      clock_type>1            ;do we want day count?
days            DS      1                       ;days count
                ENDIF
;
;************************** INTERRUPT VECTOR ****************************
;
; Note: The interrupt code must always originate at 0h.
;       A jump vector is not needed if there is no program data that needs
;       to be accessed by the IREAD instruction, or if it can all fit into
;       the lower half of page 0 with the interrupt routine.
;
                ORG     0                               ;interrupt always at 0h
;               JMP     interrupt                       ;interrupt vector
;
;************************** INTERRUPT CODE ****************************
;
; Note: Care should be taken to see that any very timing sensitive routines
;       (such as adcs, etc.) are placed before other peripherals which has
;       varying execution rates (like the software clock, for example).
;
interrupt                                               ;beginning of interrupt code
;
;****** Virtual Peripheral: Time Clock
;
; This routine maintains a real-time clock count (in msec) and allows processing
; of routines which only need to be run once every millisecond.
;
;       Input variable(s) : time_base_lo,time_base_hi,msec_lo,msec_hi
;                           seconds, minutes, hours, days
;       Output variable(s) : msec_lo,msec_hi
;                           seconds, minutes, hours, days
;       Variable(s) affected : time_base_lo,time_base_hi,msec_lo,msec_hi
;                           seconds, minutes, hours, days
;       Flag(s) affected :
;       Size : 17/39/45 bytes (depending upon clock type)
;              + 1 if bank select needed
;       Timing (turbo) : [99.9% of time] 14 cycles
;                         [0.1% of time] 17/39/45 cycles (or less)
;                        + 1 if bank select needed
;
;               BANK    clock                   ;select clock register bank
                MOV     W,#int_period           ;load period between interrupts
                ADD     time_base_lo,W          ;add it to time base
                SNC                             ;skip ahead if no underflow
                INC     time_base_hi            ;yes overflow, adjust high byte
                MOV     W,#tick_hi              ;check for 1 msec click
                MOV     W,time_base_hi-W        ;Is high byte above or equal?
                MOV     W,#tick_lo              ;load instr. count low byte
                SNZ                             ;If hi byte equal, skip ahead
                MOV     W,time_base_lo-W        ;check low byte vs. time base
                SC                              ;skip ahead if low
                JMP     :done_clock             ;If not, end clock routine
:got_tick       CLR     time_base_hi            ;Yes, adjust time_base reg.'s
                SUB     time_base_lo,#tick_lo   ; leaving time remainder
                INCSZ   msec_lo                 ;And adjust msec count
                DEC     msec_hi                 ; making sure to adjust high
                INC     msec_hi                 ; byte as necessary

                IF      clock_type>0            ;do we want sec, min, hours?
                MOV     W,#mspersec_hi          ;check for 1000 msec (1 sec tick)
                MOV     W,msec_hi-W             ;Is high byte above or equal?
                MOV     W,#mspersec_lo          ;load #1000 low byte
                SNZ                             ;If hi byte equal, skip ahead
                MOV     W,msec_lo-W             ;check low byte vs. msec count
                SC                              ;skip ahead if low
```

```
                JMP     :done_clock             ;If not, end clock routine
                INC     seconds                 ;increment seconds count
                CLR     msec_lo                 ;clear msec counters
                CLR     msec_hi                 ;
                MOV     W,#60                   ;60 seconds per minute
                MOV     W,seconds-W             ;are we at minute tick yet
                JNZ     :done_clock             ;if not, jump
                INC     minutes                 ;increment minutes count
                CLR     seconds                 ;clear seconds count
                MOV     W,#60                   ;60 minutes/hour
                MOV     W,minutes-W             ;are we at hour tick yet?
                JNZ     :done_clock             ;if not, jump
                INC     hours                   ;increment hours count
                CLR     minutes                 ;clear minutes count
                ENDIF                           ;<if> we wanted sec, min, hours

                IF      clock_type>1            ;do we want to count days?
                MOV     W,#24                   ;24 hours per day
                MOV     W,hours-W               ;are we at midnight?
                JNZ     :done_clock             ;if not, jump
                INC     days                    ;increment days count
                CLR     hours                   ;clear hours count
                ENDIF                           ;<if> we wanted day count
:done_clock
;
done_int        mov     w,#-int_period          ;interrupt every 'int_period' clocks
                retiw                           ;exit interrupt
;
;****** End of interrupt sequence
;
;************************** RESET ENTRY POINT ****************************
;
reset_entry
;               PAGE    start                   ;Set page bits and then
;               JMP     start                   ; jump to start of code
;
;*********************
;* Main Program Code *
;*********************
;
start
;               mov     !rb,#%00001111           ;Set RB in/out directions

                CLR     FSR                      ;reset all ram starting at 08h
:zero_ram       SB      FSR.4                   ;are we on low half of bank?
                SETB    FSR.3                    ;If so, don't touch regs 0-7
                CLR     IND                      ;clear using indirect addressing
                IJNZ    FSR,:zero_ram            ;repeat until done

                MOV     !OPTION,#%10011111      ;enable rtcc interrupt
;
; Main loop
;
:loop
;               BANK    Clock                   ;set clock bank
;
;       <main program code goes here>
;
                JMP     :loop                   ;back to main loop
;
;***************
;
                END                              ;End of program code
```