OLYMPIC TIMER

A Stop-Watch Timer to 1/100 Second Resolution

by
Melanie Newman
5 August 2004
melanie@tekpumps.com

A simple demonstrator in **MeLabs PICBasic Pro** showing an example of *BACKGROUND* time-keeping, using TMR1 & Interrupts *(and No embedded Assembler!)*

1. What's an Olympic Timer?

So it's Athens Olympics time (or it will be next week at the time of writing), so what better than a topical Stop-Watch Timer example that can keep time to 1/100th of a Second. Now you can challenge the official timekeeping, and when the Olympics comes to your neighbourhood, you may not have the regulation-sized swimming pool, but at least you've got the stop-watch...

It's written 100% in MeLabs PICBasic Pro, with absolutely *NO* embedded Assembler Interrupt Routines, especially for those folks that hate Assembler but can follow BASIC if it's simple enough and they've had enough beer.

2. Hardware Circuit

It's designed around a 28-pin 16F876, however, can simply be recompiled for any of the 16F87X series or 18F252 etc, and can be easily ported to suit any PIC with enough pins that will support connection of a 1 x 16 LCD (minimum requirement), and three push-buttons Yes, it can even be ported to a 16F628 or such. Because of timing limitations, this design is NOT suitable for Serial LCD connection as they are heaps slower than the usual LCD 4 or 8 wire configuration.

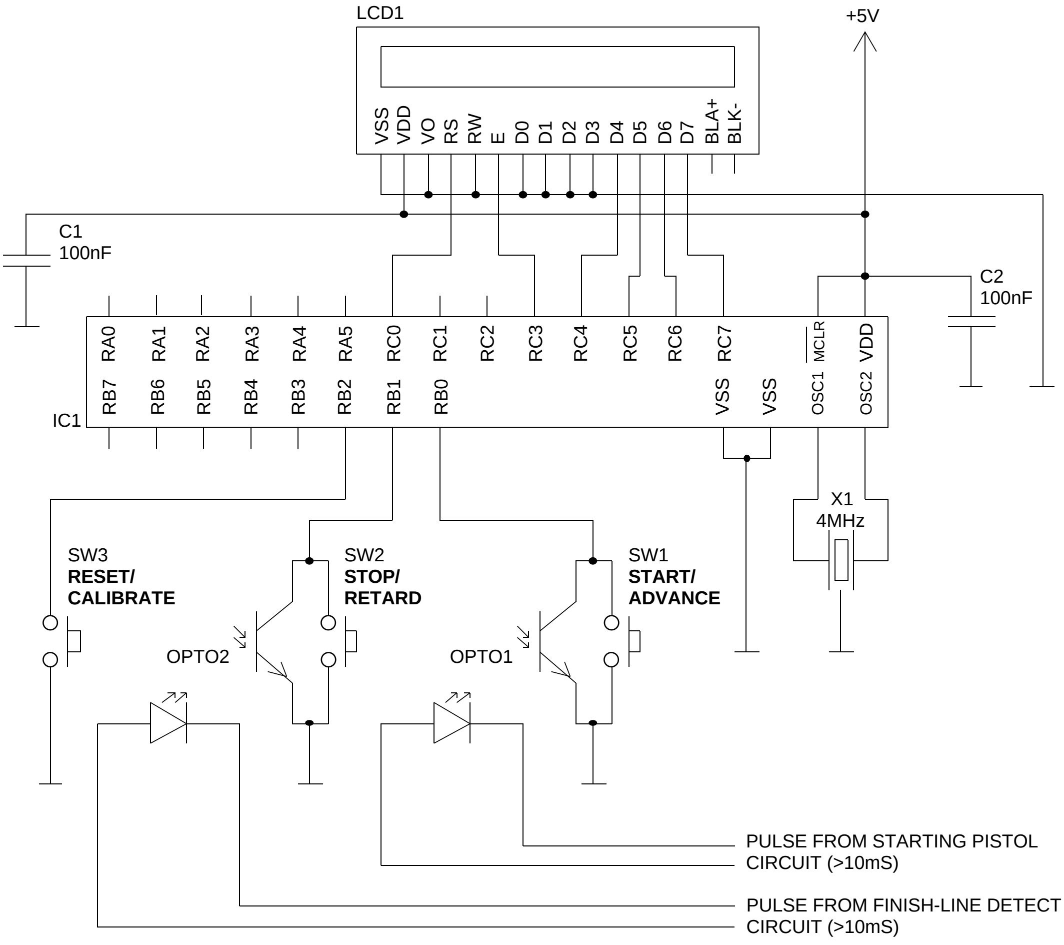
2.1 Hardware Description

LCD1 is connected to the PIC (IC1) in a classic 4-wire data arrangement. The three pushbuttons are connected to PortB to allow us to dispense with the need for any pull-up Resistors as we will be using the PIC's internal weak pull-up's. If you change the circuit and move the Buttons to any other Port, you will need to add some pull-up's. 10K should suffice.

The 16F876 needs a Crystal, Resonator or some kind of clock source (at 4Mhz for this design) to be connected. You can use a PIC with an internal oscillator as the software allows for a wide Calibration adjustment of +/- 36 Seconds in any one hour in steps as small as 360mS per Hour.

For clairty, the LCD BackLight and Contrast circuitry has been omitted.

Of course you don't need the Opto-Isolators unless you're going to be making those field Connections!



LCD1
+5V
VSS VDD VO RS RW E D0 D1 D2 D3 D4 D5 D6 D7 BLA+ BLK-
C1
100nF
C2
100nF
RA0 RA1 RA2 RA3 RA4 RA5 RC0 RC1 RC2 RC3 RC4 RC5 RC6 RC7 MCLR VDD
RB7 RB6 RB5 RB4 RB3 RB2 RB1 RB0 VSS VSS OSC1 OSC2
IC1
X1
4MHz
SW3
RESET/
CALIBRATE
SW2
STOP/
RETARD
SW1
START/
ADVANCE
OPTO2
OPTO1
PULSE FROM STARTING PISTOL
CIRCUIT (>10mS)
PULSE FROM FINISH-LINE DETECT
CIRCUIT (>10mS)

3. User Operation

Press the **START** Button (or pulse received from remote START circuit) and the Timer runs.

Press the **STOP** Button (or pulse received from remote STOP circuit) and the Timer halts.

The LCD display shows Hours, Minutes and Seconds right down to hundredths of a Second. If the Timer overflows beyond **99:59:59.99** (ie 100 Hours), then the display will roll-over back to **00:00:00.00** and continue timing with a blinking **ERR** (Error) shown on the LCD.

3.1 Timer RESET

When in STOP Mode (ie timing has halted), pressing the **RESET** Button momentarily will RESET the Timer back to all zero (00:00:00.00). The RESET Button is disabled whilst the Timer is running.

3.2 Calibration Mode

Additionally, when in STOP Mode, Pressing and HOLDING the **RESET** Button for at least FIVE SECONDS will cause the Timer to jump into it's SET-UP/CALIBRATION Mode. In this mode you can adjust the Calibration to either (+) ADVANCE (speed up) or (-) RETARD (slow down) the Timer in 1uS steps per 10mS period. This way each step adjusts 360mS to either ADVANCE or RETARD the timing per Hour. Calibration can be adjusted +/- 100 steps giving a maximum +/- 36 Second adjustment over an Hour.

When in Calibration Mode, pressing the START Button will increment the Timer Calibration Value (to ADVANCE/speed-up the Timer), whilst pressing the STOP Button will decrement the Timer Calibration Value (to RETARD/slow-down the Timer). When you have adjusted your new Calibration Value, pressing the RESET Button will **SAVE** this value and return you back into Timer Mode.

If whilst in Calibration Mode you don't press any Buttons for 20 Seconds or so, then you will automatically be reverted back to Timer Mode, **WITHOUT** any new Calibration Value being saved.

4.0 Software Description

The software operates around the BACKGROUND Operation of TMR1. This 16-bit Timer ticks every 1uS (with a 4Mhz clock) and causes an interrupt when it roll-over from $FFFF to $0000. It doesn't stop timing when this happens, it just keeps on running having set TMR1's flag (Bit 0 of the PIC's PIR1 Register). Now this roll-over and keep-timing is a neat feature that we're going to make good use of.

In an ideal world, the moment you get an interrupt flagged, you jump into your Interrupt Service Routine and do whatever has to be done. In our case we're setting an interrupt to occur every 10mS (so that we can time 1/100ths of a second). But this is not an ideal world...

In PICBasic, you don't get to jump into an Interrupt Service Routine (the routine that is specified by the ON INTERRUPT statement) until the current BASIC instruction has completed. The problem we have is that we don't know how long we are having to wait to complete our existing Basic command before we jump to the Interrupt Service Routine (ISR). It could be 1uS... or it could be several mS depending on what we're doing. The worst case scenario in our case is that complex LCDOut statement that is continually displaying the elapsed time...

```
LCDOut  $FE,$80,DEC2  Hours,":",DEC2
Minutes,":",DEC2 Seconds,".",DEC2 Hundredths
```

With all those DEC2 statements we could take several mS to execute before the ISR is actually jumped to. However, the good news is that it won't take as long as 10mS, so that's the reason we have fixed our interrupt period at 10mS to give adequate time for our longest and most time consuming PICBasic command to complete. Our main program loop therefore *CANNOT* contain any commands or instructions that would take longer than 10mS, otherwise we would miss the next interrupt tick.

Knowing that an unknown length of time has elapsed since the interrupt occurred, and before we get around to attending to it (due to the inherent latency in the way PICBasic handles interrupts), when *eventually* we DO get to service the interrupt, if we stop the Timer and read it's registers (TMR1H & TMR1L), it will actually tell us how much time has elapsed since that last interrupt rolled. We can then use this value and adjust the Timer for the next 10mS period...

For example:

If our 10mS interrupt occurred but we didn't get around to servicing it for another 3mS, then when we do reset the Timer for the next 10mS period, rather than setting it for 10mS, we set it for 7mS (10mS minus the 3mS already elapsed).

Now since TMR1 has an accuracy of 1uS (at 4Mhz), we can accurately (or at least *reasonably* accurately) dynamically adjust each following 10mS interrupt period taking into account the previously unknown PICBasic interrupt service latency. This way we can keep pretty good time.

So 10mS is $2710 (HEX), but since TMR1 counts UP from a given value thrugh $FFFF and rolls over back to $0000, we need to program TMR1 with $D8F0 ($0000-$2710) for a 10mS interrupt. Additionally I have allowed 20uS for all the hassle of stopping TMR1, reading it's previous value, and updating accordingly, so I preset TMR1 with $D910 as a starting point. Now this is by no means highly accurate, and is a 'best guess', so I have provided for a Calibration value of up to 100uS that can be added or subtracted from this 'approximate' 10mS value. This Calibration Value can be set by the user to trim the Timer exactly against some master reference. Well since all the best clocks for the last three hundred years have been provided with a calibration adjustment, I don't see why I should deviate from an entrenched habit.

This basically describes the 'core' of the program which all happens in the **SetTimer** subroutine.

The Interrupt Service Routine **TickCount** additionally just counts Hundredths of a Second, Seconds, Minutes, Hours and Overflow. The entire timekeeping function of the program is within the **TickCount** ISR. You'll notice that just because the Timer isn't running, I don't actually stop the interrupts, TMR1 runs all the time, but all that happens is that the elapsed time variables are not updated.

You'll also notice that I don't bother servicing interrupts when they are not needed (such as when you are in Reset Mode or in Set-Up/ Calibration Mode. There's no need. Furthermore, you save on lots of program space by not generating the additional code between instructions for jumping to an Interrupt routine when it is not needed.

In this case, the only salient parts of the program that end up with interrupt jumps are the dozen or so lines of the main program loop.

5.0 Questions

If you have questions, commendations or condemnations, don't email or message me off-list, please keep them on the forum...

www.picbasic.co.uk/forum

... posting them under the topic thread in the *CODE EXAMPLES* section.

6.0 PICBasic Code Listing

Starts on the next page...
Program compiles to 983 words with PBP 2.45.

```
'       Olympic Timer
'       =============
'       Melanie Newman
'       05/Aug/2004

'       Topical Program demonstrates use of Interrupt
'       Driven Background TIMER, to time events down to
'       one one-hundredth of a Second (1/100 Sec).
'
'       Bonus CALIBRATION Feature allows simple adjustments
'       in 360mS steps per hour.  This calibration adjustment
'       range is limited to +/- 36 seconds per Hour.

'
'       This program is for 4MHz clock (1uS Timer Ticks).

'
'       PIC Defines
'       ===========
'
'       Change these defines to suit your chosen PIC
'
@ DEVICE pic16F876, XT_OSC    ' System Clock Options
@ DEVICE pic16F876, WDT_ON    ' Watchdog Timer
@ DEVICE pic16F876, PWRT_ON   ' Power-On Timer
@ DEVICE pic16F876, BOD_ON    ' Brown-Out Detect
@ DEVICE pic16F876, LVP_OFF   ' Low-Voltage Programming
@ DEVICE pic16F876, CPD_OFF   ' Data Memory Code Protect
@ DEVICE pic16F876, PROTECT_OFF
                              ' Program Code Protection
@ DEVICE pic16F876, WRT_OFF   ' Flash Memory Word Enable

'
'       Hardware Defines
'       ================
        '
        '       LCD Display
        '       -----------
        '       Adjust these to suit your chosen LCD pinout
        '
Define LCD_DREG PORTC         ' Port for LCD Data
Define LCD_DBIT 4             ' Use upper 4 bits of Port
Define LCD_RSREG PORTC        ' Port for RegisterSelect (RS) bit
Define LCD_RSBIT 0            ' Port Pin for RS bit
Define LCD_EREG PORTC         ' Port for Enable (E) bit
Define LCD_EBIT 3             ' Port Pin for E bit
Define LCB_BITS 4             ' Using 4-bit bus
Define LCD_LINES 2            ' Using 2 line Display
Define LCD_COMMANDUS 2000
                              ' Command Delay (uS)
Define LCD_DATAUS 50          ' Data Delay (uS)

        '
        '       Control Buttons/Lines
        '       ---------------------
ButStart var PortB.0          ' Take this pin low momentarily to START timing
ButStop var PortB.1           ' Take this pin low momentarily to STOP timing
ButReset var PortB.2          ' Take this pin low momentarily to RESET clock
        '
        '       Hold the RESET Button pressed for at least FIVE seconds
        '       to jump into CALIBRATION Mode

'
'       Software Defines
'       ----------------
BannerOffset var BYTE         ' Variable holding start address of Banner Display
CounterA var BYTE             ' Just a Counter
CounterB var BYTE             ' Just a Counter
CounterC var BYTE
DataA var BYTE
Hours var BYTE
Hundredths var BYTE
Minutes var BYTE
OverflowError var BIT
RunningFlag var BIT
Seconds var BYTE
SetupTimeOut var WORD         ' Timeout counter for Calibration/Set-Up Mode
TMR1Cal var BYTE              ' Calibration Value
```

```
        TMR1CalAR var Byte              ' Calibration 0=ADVANCE, 1=RETARD
        TMR1RunOn var WORD              ' variable holding TMR1 Run-On value

        '
        '       EEPROM Presets
        '       --------------
        Data @0,0                       ' Advance/Retard Indicator
        Data 0                          ' Calibration Value
        Data "Olympic Timer   Powered by MeLabs PICBasic Pro"

        '
        '       Software Constants
        '       ------------------
        TMR1CalMax con 100              ' Maximum adjustment (+/-100uS per 10mS interrupt)
        TMR1Preset con $D910            ' 10mS Timer Reload value, offset by 20uS
                                        ' to allow for TMR1 Setting Calculations

        '
        '       Start Program
        '       =============

                '
                '       Initialise Processor
                '       --------------------
        TRISA=%00000000
        TRISB=%00000111
        TRISC=%00000000
        ADCON0=%11000000
        ADCON1=%00000111
        OPTION_REG.7=0                  ' Enable Pull-Up's
        RunningFlag=0                   ' Disable actual Interrupt Time-Keeping
        Pause 1000                      ' Pause for LCD to initialise
                        '
                        '       Silly Intro Banner just for Fun
                        '       -------------------------------
        LCDOut $FE,1                    ' Clear LCD
        BannerOffset=2:Gosub DisplayBanner
        Pause 2000
        For CounterA=0 to 30
                BannerOffset=2+CounterA
                Gosub DisplayBanner
                Pause 150
                Next CounterA
        Pause 1000
                '
                '       Initialise TMR1 Interrupts
                '       --------------------------
        Gosub SetTimer                  ' Set the Timer for next 10mS Interrupt
        On Interrupt goto TickCount
        PIE1.0=1                        ' Enable TMR1 Interrupts
        INTCON.6=1                      ' Enable all unmasked Interrupts
        INTCON.7=1                      ' Enable Global Interrupts
                '
                '       -----------------------------------------------------
                '       Following the above "On Interrupt", no Basic Command
                '       is allowed that takes more than 10mS to execute
                '       otherwise the 10mS Interrupt interval is compromised.
                '       -----------------------------------------------------
                '
                '       Reset Timer Variables for Start
                '       -------------------------------
DisplayReset:
        LCDOut $FE,1                    ' Clear LCD
        Read 0,TMR1CalAR                ' Read Calibration Advance/Retard Indicator
        Read 1,TMR1Cal                  ' Read Calibration Value
        Hundredths=0                    ' Reset Timer Counter variables
        Seconds=0
        Minutes=0
        Hours=0
        OverflowError=0
        '
        '       Main Program Loop
        '       =================
        Enable
DisplayLoop:
        If ButStart=0 then RunningFlag=1
        If ButStop=0 then RunningFlag=0
        LCDOut $FE,$80,DEC2 Hours,":",DEC2 Minutes,":",DEC2 Seconds,".",DEC2 Hundredths
        If OverflowError=1 then
                If Seconds.0=1 then
                        LCDOut $FE,$8C,"ERR"
                        else
                        LCDOut $FE,$8C,"   "
                        endif
                endif
        If RunningFlag=1 then goto DisplayLoop
        If ButReset=1 then goto DisplayLoop
        Disable
        '
        '       Reset Clock
        '       ===========
        '       Momentarily Press the Reset Button for RESET action.
        '       Continue holding the Reset Button for MORE than 5 seconds
        '       to jump into Calibration/Set-Up Mode
        '
ResetClock:
        LCDOut $FE,1,"Reset OK"
        Pause 1000
        Seconds=1
        While Seconds < 5
                Pause 1000
                If ButReset=1 then goto DisplayReset
                Seconds=Seconds+1
                Wend
        '
        '       Calibration Adjustment
        '       ======================
        '       If No Button is Pressed for 20 Seconds, then the program
        '       will automatically exit Calibration/Set-Up Mode WITHOUT saving
        '       any new values.
        '
        SetUpTimeout=0
Calibration:
        LCDOut $FE,1,"Calibrate: "
        While ButReset=0:Wend           ' Wait for User to release finger
CalibrationLoop:
        LCDOut $FE,$8B
        If TMR1Cal=0 then
                LCDOut " "
                else
                If TMR1CalAR=0 then
                        LCDOut "+"
                        else
                        LCDOut "-"
                        endif
                endif
        LCDOut #TMR1Cal," "
                '       ----------------------------------------------------------
                '       Press Start Button to ADVANCE (speed-up) Clock
                '       Press STOP Button to RETARD (slow-down) Clock
                '       Press RESET Button to SAVE new Calibration Setting
                '       ----------------------------------------------------------
                '       Remember each Calibration 'tick' will advance or
                '       retard the Timing by 1uS in every 10mS period - that's
                '       360mS/Hour per setting.  Example:  A setting of +8 will
                '       SPEED-UP the Timer by 2.88 Seconds (8 x 360mS) in an Hour.
                '       ----------------------------------------------------------
        If TMR1CalAR=0 then
                If ButStart=0 then Gosub CalAdvance
                If ButStop=0 then Gosub CalRetard
                else
                If ButStart=0 then Gosub CalRetard
                If ButStop=0 then Gosub CalAdvance
                endif
        If ButReset=0 then
                Write 0,TMR1CalAR
                Write 1,TMR1Cal
                LCDOut $FE,1,"Have a Nice Day"
                Pause 1000
                Goto DisplayReset
                endif
        SetupTimeout=SetupTimeout+1
        If SetupTimeout>200 then goto DisplayReset
        Pause 100
        Goto CalibrationLoop
        '
        '       Subroutine Increments Calibration Value
        '       ---------------------------------------
CalAdvance:
        SetupTimeout=0
        If TMR1Cal=>TMR1CalMax then
                TMR1Cal=TMR1cALmAX
                TMR1CalAR=TMR1CalAR^1
                else
                TMR1Cal=TMR1Cal+1
                endif
        Return

        '
        '       Subroutine Decrements Calibration Value
        '       ---------------------------------------
CalRetard:
        SetupTimeout=0
        If TMR1Cal=0 then
                TMR1Cal=1
                TMR1CalAR=TMR1CalAR^1
                else
                TMR1Cal=TMR1Cal-1
                endif
        Return

        '
        '       Subroutine Displays Banner Intro
        '       --------------------------------
DisplayBanner:
        CounterC=BannerOffset+15
        LCDOut $FE,$80
        For CounterB=BannerOffset to CounterC
                Read CounterB,DataA
                LCDOut DataA
                Next CounterB
        Return

        '
        '       Subroutine Loads TMR1 values
        '       ============================
SetTimer:
        T1CON.0=0                       ' Stop the Clock
        TMR1RunOn.Highbyte=TMR1H        ' Load the Run-On (Over-Run) value (if any)
        TMR1RunOn.Lowbyte=TMR1L
        TMR1RunOn=TMR1Preset+TMR1RunOn
                                        ' Calculate the New (adjusted) value for TMR1
        If TMR1CalAR=0 then             ' Calibration ADVANCE (add) or RETARD (subtract)
                TMR1RunOn=TMR1RunOn+TMR1Cal
                else
                TMR1RunOn=TMR1RunOn-TMR1Cal
                endif
        TMR1H=TMR1RunOn.Highbyte        ' Save new values to TMR1
        TMR1L=TMR1RunOn.Lowbyte
        T1CON.0=1                       ' Restart the Clock
        PIR1.0=0                        ' Reset TMR1's Interupt Flag
        Return

        '
        '       Timer Interrupt Handler
        '       =======================
TickCount:
        Gosub SetTimer                  ' Set the Timer for next 10mS Interrupt
        If RunningFlag=1 then           ' If timing actually enabled... then...
                Hundredths=Hundredths+1
                                        ' Increment 10mS Seconds Counter
                If Hundredths>99 then
                        Hundredths=0
                        Seconds=Seconds+1
                                        ' Increment the Seconds
                        If Seconds>59 then
                                Seconds=0
                                Minutes=Minutes+1
                                        ' Increment the Minutes
                                If Minutes>59 then
                                        Minutes=0
                                        Hours=Hours+1
                                        ' Increment the Hours
                                        If Hours>99 then
                                        ' Handle any Overflow
                                                Hours=0
                                                OverFlowError=1
                                                endif
                                        endif
                                endif
                        endif
                endif
        Resume

        End
```