INTRODUCTION

Recently, Microchip has introduced a series of new PIC16 and PIC18 PICmicro® microcontrollers that have an on-board Internal RC Oscillator, capable of eight frequencies from 31 kHz to 8 MHz. These microcontrollers include the following devices:

- PIC16F818/819
- PIC16F87/88
- PIC18F1X20
- PIC18F2220/2320/4220/4320

The internal RC oscillator is configured and tuned by the OSCCON and OSCTUNE registers, respectively. The OSCCON register sets the frequency and contains the IOFS bit, which indicates when the frequency has stabilized after a frequency change in the OSCCON register. OSCTUNE is responsible for tuning the frequency within a range of +/-12 percent on the selected frequency. The upper range of tuning is 01h - 1Fh and the lower range is 20h - 3Fh, with center frequency at 00h. It is recommended to consult the device data sheet for further details.

The assembly code included in this application note (see Appendix A) is only applicable for a PIC16F819 microcontroller; however, with minor changes, the code can be ported over to the devices listed above. Having an internal RC oscillator eliminates the need for capacitors and an external crystal, resulting in production cost savings.

You may be wondering, “Why would it be necessary to calibrate the internal oscillator if it is supposed to come from the factory already calibrated?” Even though the internal oscillator is factory calibrated at 25°C, temperature variations in the operating environment must be considered. Since a change in frequency, as little as two to three percent, can corrupt serial communication, the information presented in this application note can help you avoid this issue by allowing frequency calibration over temperature.

The ability to calibrate the internal oscillator means that a known time-base must be established as a reference point. This time-base will be derived from a 32.768 kHz clock crystal connected to Timer1 (see Figure 1). Other fixed time-bases that can be used are AC line frequency (50 or 60 Hz).

FIGURE 1: TIMER1 OSCILLATOR WITH 32.768 KHZ XTAL

In this application note, we will be calibrating the 4 MHz tap of the internal oscillator. The calibration method will compare the time it takes for a specific number of instructions to occur over a defined period of Timer1. It is not mandatory to calibrate the internal oscillator at 4 MHz as Table 1 indicates; however, if you choose 4 MHz for calibration and your code requires a frequency change after calibration is complete, the other frequency chosen will be calibrated.
TECHNICAL DETAILS

Before going further, let us first establish several system numbers that will be important for understanding the techniques used in this application note. The values in Table 1 were derived from the base numbers shown below in Figure 1.

Knowing that 30.5 instructions can be executed within one Timer1 count, we can now determine if the oscillator requires calibration and whether it is too fast or too slow.

There are several variables that are required for this calibration approach. The first is the number of Timer1 counts. This number is subtracted from FFFFh to determine what value TMR1H:TMR1L will be loaded with. This is not a code-defined variable, but you may choose to define it as such. A defined constant, called calc_instr, is set by the user for the expected number of instructions that will be executed for a defined Timer1 interval established by the TMR1H:TMR1L value.

The second defined constant is f_delta_calc, which is set by the user according to their accuracy requirements. This number is the allowable difference between the calculated number of instructions expected (calc_instr) and the measured (mesrd_instr) number of instructions executed.

As previously mentioned, Table 1 shows several of the available internal RC oscillator frequencies. For each of these frequencies, there are instruction execution times with +/-1% accuracy, and the calculated number of instructions for 25 Timer1 counts. The mesrd_instr variable increments until the Timer1 overflow flag is set. The f_delta_count, as mentioned previously, is the acceptable difference by mesrd_inst and calc_instr. By changing the value shown in the T1 CNTS column of Table 1, the following columns will yield different results. If the f_delta_count is increased, the calibration time is decreased, but accuracy will fall off.

<table>
<thead>
<tr>
<th>INTRC Freq</th>
<th>Fosc/4</th>
<th>Tcy (µs)</th>
<th>Tcy/T1 Count</th>
<th>Drift (%)</th>
<th>Freq (-1%)</th>
<th>Tcy (µs) (Slow)</th>
<th>Tcy/T1 Count</th>
<th>Freq (+1%)</th>
<th>Tcy (µs) (Fast)</th>
<th>Tcy/T1 Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000000</td>
<td>2000000</td>
<td>0.500</td>
<td>61.036</td>
<td>1</td>
<td>1980000</td>
<td>0.505</td>
<td>60.426</td>
<td>2020000</td>
<td>0.495</td>
<td>61.646</td>
</tr>
<tr>
<td>4000000</td>
<td>1000000</td>
<td>1.000</td>
<td>30.518</td>
<td>1</td>
<td>990000</td>
<td>1.010</td>
<td>30.213</td>
<td>1010000</td>
<td>0.990</td>
<td>30.823</td>
</tr>
<tr>
<td>2000000</td>
<td>500000</td>
<td>2.000</td>
<td>15.259</td>
<td>1</td>
<td>495000</td>
<td>2.020</td>
<td>15.106</td>
<td>505000</td>
<td>1.980</td>
<td>15.412</td>
</tr>
<tr>
<td>1000000</td>
<td>250000</td>
<td>4.000</td>
<td>7.630</td>
<td>1</td>
<td>247500</td>
<td>4.040</td>
<td>7.553</td>
<td>252500</td>
<td>3.960</td>
<td>7.706</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T1 CNTS</th>
<th>Tcy Total (Slow)</th>
<th>Tcy Total (Fast)</th>
<th>calc_instr</th>
<th>f_delta_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>1510.6</td>
<td>1525.90</td>
<td>381.5</td>
<td>3.8</td>
</tr>
<tr>
<td>25</td>
<td>755.3</td>
<td>762.95</td>
<td>190.7</td>
<td>1.9</td>
</tr>
<tr>
<td>25</td>
<td>377.7</td>
<td>381.48</td>
<td>95.4</td>
<td>1.0</td>
</tr>
<tr>
<td>25</td>
<td>188.8</td>
<td>190.74</td>
<td>47.7</td>
<td>0.5</td>
</tr>
</tbody>
</table>

EXAMPLE 1: CALIBRATION BASE NUMBERS

- Timer1 Clock Crystal: 32.768 kHz → 1/32,768 = 30.5 µs period
- System Clock: 4 MHz → Fosc/4 = 1 MHz core clock = 1 µs Instruction time (Tcy)
- 4 MHz @ +1% = 1.01 MHz core clock = .990 µs Tcy
- 4 MHz @ -1% = .990 MHz core clock = 1.01 µs Tcy
- @ 4 MHz (1 µs Tcy), 30.5 core instructions can be executed for one count of Timer1.
CODE DESCRIPTION

Before the calibration code is executed as shown in the flow diagram in Figure 2, the user must ensure that the 32.768 kHz clock crystal connected to Timer1 has adequate time to become stable. If this stability is not ensured, then the calibration routine will be driven from an inaccurate time-base. The required initialization is minimal; the internal oscillator is configured for 4 MHz via the OSCCON register. Timer1 is configured via T1CON for:

- 1:1 prescale
- Osc enabled
- Do not synchronize external clock input
- External clock from T1OSO
- Timer1 enabled

Upon entry into the calibration routine, the TMR1H:TMR1L registers are loaded with 0xFFE7h, which establishes 25 Timer1 counts until overflow. An incrementing loop is entered and exit only takes place when a Timer1 overflow occurs (TMR1IF = 1).

Next, the mesrd_instr variable is subtracted from the calc_instr constant to determine if the frequency is slower or faster than 4 MHz. Once this decision is made by a bit test on the carry flag, either the slower_freq or faster_freq subroutine will be entered. In both of these instruction sets, the current frequency is checked if it is acceptable for operation. This is determined by comparing the difference between mesrd_instr and calc_instr with the value of f_delta_count. If the result is acceptable, then the calibration is complete. If the result is unacceptable, the OSCTUNE register decrements or increments, depending on the speed of the measured frequency.

When the OSCTUNE register is modified, the new “tuned” values take approximately 256 µs to become stable. Therefore, a delay (tune_delay) must be called before the frequency can be checked again. When the delay_temp variable is loaded with 0x57, the tune_delay loop will take up to 266 µs @ 4 MHz.

FIGURE 2:  CALIBRATION PROCESS FLOW DIAGRAM

![Flow Diagram](image-url)
CONCLUSION

The assembly code has been written as relocatable code so that it can easily be added to an existing application. The code usage is 56 words of program memory, 2 bytes of RAM and takes approximately 31 ms from start to finish. Although specific numbers have been used in this application, they may be different for other application requirements. TMR1H:TMR1L can be loaded with different values, which can speed up the calibration routine with less frequency sampling, or slow the routine with more sampling.

The \textit{f\_delta\_count} value in this application is defined for $\pm 1\%$, but can be changed for 2 or 3 percent accuracy. These variables can be used in applications at the discretion of the user. Some applications may require that the OSCTUNE value after calibration be retained for RESET or POR conditions. This can be accomplished by storing the value in the PICmicro microcontroller's on-board EEPROM.

So that an organized starting point in your code development can be established, it is recommended that you create a spreadsheet file similar to that of Table 1. Doing this will also help in fine tuning the calibration boundaries to the application.
APPENDIX A: SOURCE CODE

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") is intended and supplied to you, the Company's customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

;*******************************************************************************
;* This code assumes the following conditions:
;*
;* 1. Config bits FOSC2:FOSC0 set to INTRC.
;*
;* 2. INTOSC is configured for 4MHz system clock via OSCCON reg.
;*
;* 3. Timer1 has been configured for 32.768kHz external crystal
;*    and stable operation has been established.
;*
;* Call from main: call int_rc_cal
;*******************************************************************************

;* File Name: rc_cal.asm
;* Dependencies: P16F819.INC, 16f819.1kr
;* Processor: 16F819
;* MPLAB IDE 6.10.0.0
;* MPASM Tool Suite 1.10.0.0
;* Company: Microchip Technology, Inc.
;* Software License Agreement
;*
;* The software supplied herewith by Microchip Technology Incorporated
;* (the "Company") for its PICmicro® Microcontroller is intended and
;* supplied to you, the Company's customer, for use solely and
;* exclusively on Microchip PICmicro Microcontroller products. The
;* software is owned by the Company and/or its supplier, and is
;* protected under applicable copyright laws. All rights are reserved.
;* Any use in violation of the foregoing restrictions may subject the
;* user to criminal sanctions under applicable laws, as well as to
;* civil liability for the breach of the terms and conditions of this
;* license.
;*
;* THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
;* WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
;* TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
;* PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
;* IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
;* CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;*
;* Author Date Comment
;*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;* Chris Valenti Nov 19, 2002 Initial Release (V1.0)
;*
;*******************************************************************************

list P=16F819
#include <P16F819.INC>
errorlevel -302

;************************ Defined constants *******************************
#define calc_instr .190 ; calculated # of instructions
#define f_delta_calc .1 ;acceptable difference

;********************************************************************
;********************** Variables ***********************************
cal_var UDATA_SHR 0x20
mesrd_instr RES 1 ;measured # of instructions executed
delay_temp RES 1 ;delay variable
;********************************************************************
cal CODE 0x0005
;******* Internal RC Calibration Routine ********************
int_rc_cal

banksel TMR1H
bcf PIR1,TMR1IF ;clear TIMER 1 interrupt flag
clf mesrd_instr ;clear loop count register
movlw 0xFF ;use 25 counts of Timer 1
movwf TMR1H
movlw 0xE7
movwf TMR1L ;T1 starts count here
incf mesrd_instr,f ;increment
btfss PIR1,TMR1IF ;has TMR1 overflowed ?
goto $-2 ;NO, continue incrementing

count_done ;YES, start calibration
banksel mesrd_instr
movf mesrd_instr,w
sublw calc_instr ;calc_instr - mesrd_instr, find f_delta
btfs STATUS,C ;Is the Frequency slow or fast?
goto slower_freq ;mesrd_count > calc_count, freq is fast
b goto faster_freq ;mesrd_count < calc_count, freq is slow

slower_freq ;DECREASE FREQUENCY
sublw 0xFF ;FF - w_reg
sublw f_delta_calc ;Check if freq is acceptable
btfsc STATUS,C ;Is w_reg > f_delta_calc?
goto done ;NO, calibration done!
banksel OSCTUNE
decf OSCTUNE,F ;YES, decrease the frequency
call tune_delay ;allow OSCTUNE change to take effect
goto int_rc_cal ;calibrate again

faster_freq ;INCREASE FREQUENCY
sublw f_delta_calc ;Check if freq is acceptable
btfs STATUS,C ;Is w_reg > f_delta_calc?
goto done ;NO, calibration done!
banksel OSCTUNE
incf OSCTUNE,f ;YES, increase the frequency
call tune_delay ;allow OSCTUNE to take effect
goto int_rc_cal ;calibrate again
done

return ;continue with main code
GLOBAL int_rc_cal

;*************************************************************************
; Function: tune_delay
;
; Overview:
; This routine is used as a delay to allow the tuned value of
; OSCTUNE to take effect. >87 loaded into delay_temp will yield
; a 266us delay @ 4MHz
;
; Input:   None
; Output:  None


; Stack Requirements: 1 level deep
;
; Side Effects: Databank changed
;*************************************************************************

;clock stabilization delay (8 clocks)

    banksel delay_temp
    movlw .87
    movwf delay_temp
    decfsz delay_temp,f
    goto $-1
    return

end

Section Info

<table>
<thead>
<tr>
<th>Section</th>
<th>Type</th>
<th>Address</th>
<th>Location</th>
<th>Size(Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STARTUP</td>
<td>code</td>
<td>0x000000</td>
<td>program</td>
<td>0x00000a</td>
</tr>
<tr>
<td>cal</td>
<td>code</td>
<td>0x000005</td>
<td>program</td>
<td>0x000056</td>
</tr>
<tr>
<td>.cinit</td>
<td>romdata</td>
<td>0x000030</td>
<td>program</td>
<td>0x000004</td>
</tr>
<tr>
<td>main_c</td>
<td>code</td>
<td>0x000040</td>
<td>program</td>
<td>0x00012</td>
</tr>
<tr>
<td>.config</td>
<td>code</td>
<td>0x002007</td>
<td>program</td>
<td>0x00002</td>
</tr>
<tr>
<td>cal_var</td>
<td>udata</td>
<td>0x000020</td>
<td>data</td>
<td>0x00002</td>
</tr>
</tbody>
</table>

Program Memory Usage

Start End

0x000000 0x000031
0x000040 0x000048
0x002007 0x002007

60 out of 2312 program addresses used, program memory utilization is 2%

Symbols - Sorted by Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Location</th>
<th>Storage</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>int_rc_cal</td>
<td>0x000005</td>
<td>program</td>
<td>extern</td>
<td>C:\Program Files\MPLAB IDE\rc_cal.asm</td>
</tr>
<tr>
<td>_cal_000F</td>
<td>0x00000f</td>
<td>program</td>
<td>static</td>
<td>C:\Program Files\MPLAB IDE\rc_cal.asm</td>
</tr>
<tr>
<td>count_done</td>
<td>0x000010</td>
<td>program</td>
<td>static</td>
<td>C:\Program Files\MPLAB IDE\rc_cal.asm</td>
</tr>
<tr>
<td>slower_freq</td>
<td>0x000017</td>
<td>program</td>
<td>static</td>
<td>C:\Program Files\MPLAB IDE\rc_cal.asm</td>
</tr>
<tr>
<td>faster_freq</td>
<td>0x000020</td>
<td>program</td>
<td>static</td>
<td>C:\Program Files\MPLAB IDE\rc_cal.asm</td>
</tr>
<tr>
<td>done</td>
<td>0x000028</td>
<td>program</td>
<td>static</td>
<td>C:\Program Files\MPLAB IDE\rc_cal.asm</td>
</tr>
<tr>
<td>tune_delay</td>
<td>0x000029</td>
<td>program</td>
<td>static</td>
<td>C:\Program Files\MPLAB IDE\rc_cal.asm</td>
</tr>
<tr>
<td>_cal_002E</td>
<td>0x00002e</td>
<td>program</td>
<td>static</td>
<td>C:\Program Files\MPLAB IDE\rc_cal.asm</td>
</tr>
<tr>
<td>Start</td>
<td>0x000040</td>
<td>program</td>
<td>static</td>
<td>C:\Program Files\MPLAB IDE\main.asm</td>
</tr>
<tr>
<td>mesrd_instr</td>
<td>0x000020</td>
<td>data</td>
<td>static</td>
<td>C:\Program Files\MPLAB IDE\rc_cal.asm</td>
</tr>
<tr>
<td>delay_temp</td>
<td>0x000021</td>
<td>data</td>
<td>static</td>
<td>C:\Program Files\MPLAB IDE\rc_cal.asm</td>
</tr>
</tbody>
</table>
Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip’s Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip’s products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Keeloq, MPLAB, PIC, PICmicro, PICSTART, PRO MATE and PowerSmart are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

FilterLab, microID, MXDEV, MXLAB, PICMASTER, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Accuron, dsPIC, dsPICDEM.net, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, PICC, PICkit, PICDEM, PICDEM.net, PowerCal, PowerInfo, PowerTool, rPIC, Select Mode, SmartSensor, SmartShunt, SmartTel and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2003, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999 and Mountain View, California in March 2002. The Company’s quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KeeloQ® code hopping devices, Serial EEPROMs, microperipherals, non-volatile memory and analog products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001 certified.