

Figure 1: Schematic diagram showing how the MAX6576 temperature sensor can be connected to a PIC16F884 microcontroller.

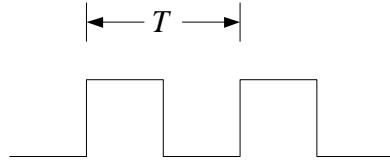


Figure 2: Waveform from the MAX6576 temperature sensor. The period  $T$  is proportional to the temperature. The duty cycle of the wave is 50%.

## 1 Using the PIC16F884 Timer 1 to Determine the Temperature

The Maxim MAX6576 is a temperature sensor whose output is a square wave with a period that is proportional to the temperature. This can be connected to a PIC16F884 microcontroller which can then deduce the temperature by using Timer 1 to measure the period of each wave received from the sensor.

A schematic diagram showing how the two circuits can be connected together is shown in Figure 1.

The datasheet for the MAX6576 shows that the output of the sensor looks like the waveform shown in Figure 2.

The signals  $TS_1$  and  $TS_0$  control the scale factor  $k_\tau$  which relates the

TS1	TS0	$\frac{k_\tau}{\mu\text{s}/\text{K}}$
0	0	10
0	1	40
1	0	160
1	1	640

Table 1: Scale factors available with the MAX6576.

temperature  $\tau$  to the period  $T$ .

Figure 1 on the preceding page uses a setting of TS1= 1 and TS0= 0, corresponding to a scale factor of  $k_\tau = 160 \mu\text{s}/\text{K}$ . If we measure the period of the sensor’s output wave to be  $T_{\text{MEASURED}}$ , then we can find the temperature in °C using this equation:

$$\begin{aligned}\tau &= \frac{T_{\text{MEASURED}}}{k_\tau} - 273.15 \text{ K} \\ \tau &= \frac{T_{\text{MEASURED}}}{160 \mu\text{s}/\text{K}} - 273.15 \text{ K.}\end{aligned}\quad (1)$$

For example, if the measured period were  $T_{\text{MEASURED}} = 48 \text{ ms}$ , then the temperature would be 26.85 °C. The MAX6756 typically only gives an accuracy of  $\pm 0.8 \text{ }^\circ\text{C}$ , so rounding off this result to 27 °C is appropriate. This corresponds to the outside temperature on a warm summer day, roughly 80 °F.

The circuit depicted in Figure 1 on the previous page shows the square wave emitted by the MAX6576 temperature sensor being applied to the INT input of the PIC16F884. This is a convenient place to apply the signal because it permits the microcontroller to be interrupted every time the square wave goes high. If we read the value of the 16-bit counter, Timer 1, at the start of the interrupt service routine and then clear the counter, we will get rubbish the very first time the INT interrupt occurs but, thereafter, the Timer 1 value will show the number of input clocks it received between successive interrupts. This will permit calculating the temperature.

Let’s assume that the PIC16F884 is being driven with an external crystal whose frequency is  $f_{\text{osc}} = 10 \text{ MHz}$ , corresponding to an instruction rate of  $f_{\text{INST}} = 2.5 \text{ MHz}$ , or an instruction period of 400 ns. The example above showed that we need to be able to measure intervals on the order

Prescale	
Ratio	Count
1:1	125 000
1:2	62 500
1:4	31 250
1:8	15 625

Table 2: The Timer 1 prescaler can reduce the values that need to be stored in the 16-bit register.

of  $T_{\text{MEASURED}} = 50 \text{ ms}$  in order to measure temperatures a little above room temperature ( $20^\circ\text{C}$ ).

Let's figure out how many instruction cycles this corresponds to:

$$\begin{aligned}
 n &= \frac{T_{\text{MEASURED}}}{T_{\text{INST}}} \\
 &= T_{\text{MEASURED}} f_{\text{INST}} \\
 &= T_{\text{MEASURED}} \frac{f_{\text{OSC}}}{4} \\
 &= (50 \text{ ms}) \left( \frac{10 \text{ MHz}}{4} \right) \\
 &= 125 000.
 \end{aligned}$$

This number is too big to fit within the eight-bit counter of Timer 1. We can elect to apply prescale ratios of 1:1, 1:2, 1:4, or 1:8. Table 2 shows the reduced values we would get with each choice for the prescalar. The 1:4 ratio gives us 31 250, a number that is nearly in the middle of the range a 16-bit counter can hold, so we'll pick that value. If we needed greater accuracy, we might wish to pick a smaller prescale value. Here, only the ratio 1:2 would do this, yet still provide a number that would fit within the 16-bit register. However, it would place an upper limit on the temperature of around  $55^\circ\text{C}$ .

With the prescalar in use and set to the 1:4 ratio, we can calculate the measured time from the count  $n$  in Timer 1 as

$$\begin{aligned}
 T_{\text{MEASURED}} &= 4nT_{\text{INST}} \\
 &= 4n (400 \text{ ns}) \\
 &= 1.6n \mu\text{s}.
 \end{aligned}$$

Substituting this into (1) gives

$$\begin{aligned}\tau &= \frac{T_{\text{MEASURED}}}{160 \text{ }\mu\text{s/K}} - 273.15 \text{ K} \\ &= \frac{1.6n \text{ }\mu\text{s}}{160 \text{ }\mu\text{s/K}} - 273.15 \text{ K} \\ &= \left( \frac{n}{100} - 273.15 \right) \text{ K.}\end{aligned}$$

This equation applies more precision than the sensor is capable of, so we should round the result off to the nearest degree.

$$\tau = \left( \frac{n}{100} - 273 \right) \text{ K.} \quad (2)$$

To carry out the division, it is useful to have a routine for dividing a 16-bit number from Timer 1 by an 8-bit number, which is large enough to hold the divisor, 100.

The listing shown at the end of this document includes a subroutine called **Divide**. It can divide a 16-bit dividend by an eight-bit divisor, yielding a 16-bit quotient and an eight-bit remainder. This subroutine begins on line 634 of the listing. Not only is it useful for converting the measured time read from Timer 1 into the corresponding temperature. It also is useful in extracting the decimal digits of that temperature so that they can be displayed sensibly using the seven-segment displays attached to the processor.

The circuit as drawn is not practical because it might need to sink more current through the pins driving the LEDs than the PIC processor can tolerate. If each of the 22 pins of this sort needed to sink 5 mA, then there would be a total requirement to sink 110 mA, more than the 90 mA the chip can tolerate.

So although the schematic is easy to follow, it is too simplistic. Two solutions would be easy:

- Turn on only one display at a time, cycling through them fast enough so that there is no apparent flicker. This would keep the current sink requirement below 35 mA or so.
- Use current buffers to drive the LED displays.

A **cblock** directive in lines 135–153 is used to define the variables used by the program. There are three categories of these: some are needed by

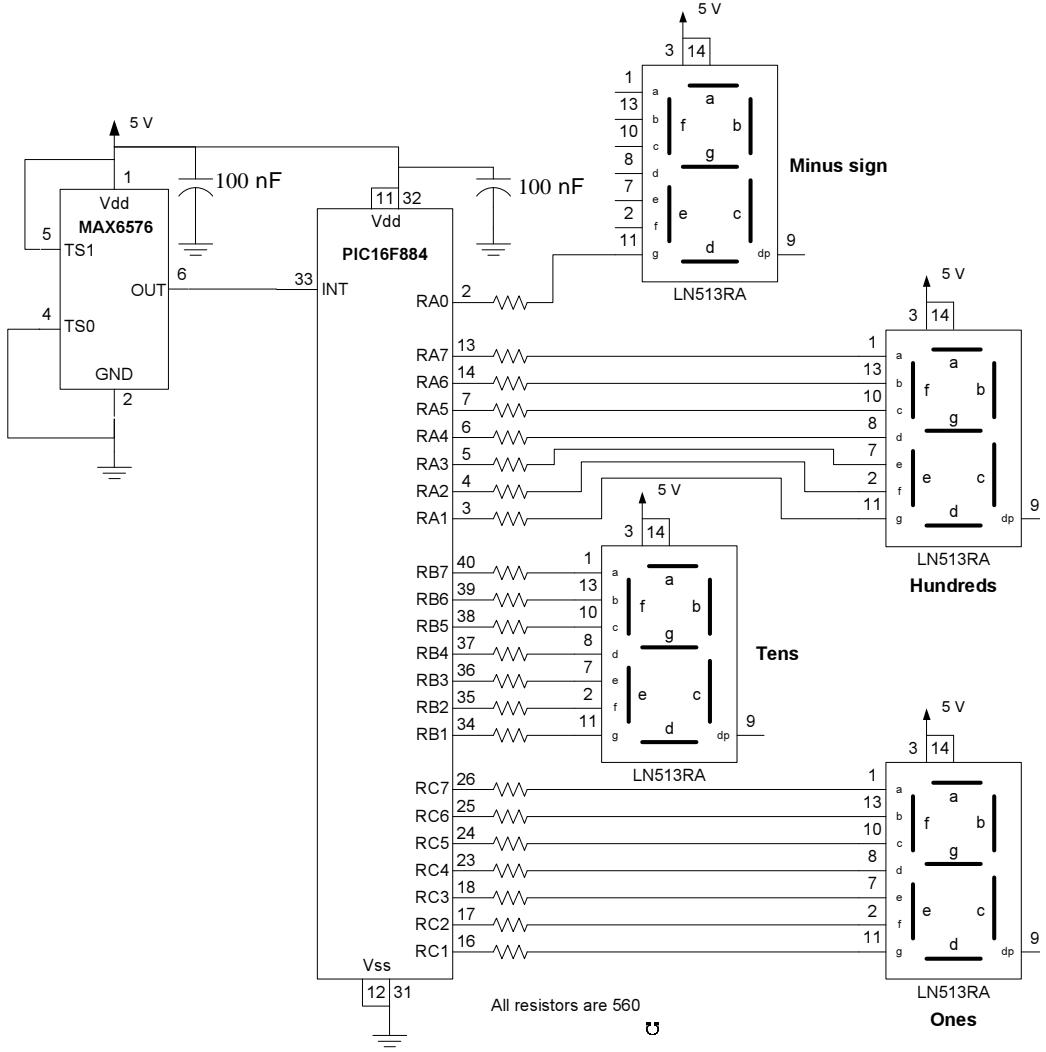


Figure 3: Full schematic for the temperature sensor. Note that as depicted here, too much current needs to be sunk by the microcontroller. A software solution would be to turn on just one display at a time, rotating through them all in a round-robin fashion in order not to violate the current limit. This might necessitate more than the 5 mA current through each diode in order to get adequate average brightness. The specified display can handle up to 20 mA through each segment, although with all seven segments running, this would exceed the PIC's limit of 90 mA.

the Divide subroutine, some to take a snapshot of the value in Timer 1, and some used by the main program.

The return codes defined in lines 157 and 158 are returned by the Divide routine to indicate its level of success. In this program, they are ignored.

The seven-segment displays all were connected to the high-order seven bits of ports A, B, and C. Positioning them all in the same way simplifies the code that puts suitable values in these display registers. One exception is the use of RA0 to operate the horizontal segment in the middle of one seven-segment display. This serves as a minus sign, used when negative temperatures are present.

Two macros are defined in lines 259–276 (for setting the bank-select bits) and in lines 284–308 (for transferring a seven-segment display pattern to a specified port.) The second of these might better have been defined as a subroutine, but the macro syntax makes it easier to invoke than the syntax for calling a subroutine, since in this assembly language, all variables need to be stored in registers before the subroutine is called.

The initialization code entails setting up the output ports and Timer 1, as well as enabling the external interrupt from the temperature sensor. The codes in lines 465–505 are devoted to converting the count from Timer 1 into a corresponding temperature, using (2). This value is loaded into the T1H:T1L register pair by the interrupt service routine whenever an external interrupt occurs. Since these are triggered by every rising edge of the signal from the sensor, we in effect convert the most recently captured value just as fast as the main program can get through its loop.

The sign of the temperature is used to decide whether or not to illuminate the negative sign in the display. The Divide routine is used three times to extract the three digits in the temperature. A table look-up subroutine is used to convert these digits to seven-segment display patterns.

The bulk of the interrupt processing is handled by the subroutine in lines 682–706. Its job is quite simple: capture the two bytes of Timer 1 and put them in a location where the main program can find them when it needs them.

There is one tricky point: the high-order byte might roll over between the time it is read and the time the low-order byte is read. To compensate for this, it is read a second time. If there has been a change, both bytes are read again.

The main program can be seen to be responsible for two main things:

- Convert timer values to corresponding temperatures.
- Display the current temperature.

The interrupt service routines sole job is to record the value of the timer every time the temperature sensor has taken a measurement and to zeroize the timer again in preparation for another measurement.

4 LOC OBJECT CODE LINE SOURCE TEXT  
5 VALUE

6

7       00001 ;\*\*\*\*\*  
8       00002 ; This file includes a program for reading the outputs of a Maxim \*  
9       00003 ; 6576 temperature sensor and deducing the temperature from the \*  
10      00004 ; reading. \*  
11      00005 ; \*  
12      00006 ;\*\*\*\*\*  
13      00007 ; \*  
14      00008 ;     Filename: TemperatureSensor.asm \*  
15      00009 ;     Date: September 25, 2008 \*  
16      00010 ;     File Version: 1 \*  
17      00011 ; \*  
18      00012 ;     Author: CDR Charles B. Cameron \*  
19      00013 ;     Company: United States Naval Academy \*  
20      00014 ; \*  
21      00015 ; \*  
22      00016 ;\*\*\*\*\*  
23      00017 ; \*  
24      00018 ;     Files required: \*  
25      00019 ;                   p16f884.inc    Predefined variables to make code \*  
26      00020 ;    readable. \*  
27      00021 ; \*  
28      00022 ;\*\*\*\*\*  
29      00023 ; \*  
30      00024 ;     Notes: \*  
31      00025 ;     Use the RB0/INT input to get data from the MAX6576. \*  
32      00026 ;     Enable interrupts so that the contents of Timer 0 can be \*  
33      00027 ;     read and the Timer can be cleared to zero. This will cause \*  
34      00028 ;     Timer 0 to indicate the number of counts that have elapsed \*  
35      00029 ;     between successive interrupts. Knowing the time each count \*  
36      00030 ;     takes allows the period to be deduced. Knowing the relation- \*  
37      00031 ;     ship between this period and the temperature lets the temper- \*  
38      00032 ;     ature be deduced. \*  
39      00033 ; \*  
40      00034 ;     This program uses the subroutine Divide, which needs the \*  
41      00035 ;     following registers: \*  
42      00036 ; \*  
43      00037 ;     The 16-bit dividend D gets divided by the 8-bit \*  
44      00038 ;     divisor V to form a 16-bit quotient Q and an \*  
45      00039 ;     8-bit remainder R. \*  
46      00040 ; \*  
47      00041 ;     Register usage for the subroutine \*  
48      00042 ;     DU    An 8-bit extension of the dividend \*  
49      00043 ;     used internally by the subroutine for \*  
50      00044 ;     doing arithmetic \*  
51      00045 ;     DH:DL MSB and LSB of Dividend D \*  
52      00046 ;     VL    Divisor V \*  
53      00047 ;     QH:QL Quotient Q \*  
54      00048 ;     RL    Remainder R \*  
55      00049 ;     Count A counter needed to control the number of \*  
56      00050 ;     times the subroutine loops. \*  
57      00051 ;     W     On exit, holds an error code. \*  
58      00052 ; \*  
59      00053 ;     To use the subroutine, load the dividend and the divisor \*

```

61
62
63 LOC OBJECT CODE LINE SOURCE TEXT
64 VALUE
65
66          00054 ; registers , then call the subroutine. On exit , the W register      *
67          00055 ; indicates the success of the division. It will contain the      *
68          00056 ; error code DivideByZero if the divisor was a zero. It will      *
69          00057 ; contain the error code SuccessfulDivision otherwise      *
70          00058 ;
71          00059 ;
72          00060 ;*****                                                 *
73          00061
74          00062
75          00063 list      p=16f884           ; list directive to define processor
76          00064 #include <p16f884.inc>       ; Processor-specific variable
77          00001 LIST
78          00002 ; P16F884.INC Standard Header File , Version 1.00 Microchip Technology , Inc.
79          000618 LIST
80          00065 ; definitions. These define the
81          00066 ; mnemonic symbols used in the
82          00067 ; data sheet , such as PORTA, F, and W.
83          00068
84          00069 ; The following sequence determines the value of the configuration
85          00070 ; word and then uses it to set the configuration. Additional options
86          00071 ; are specified in each successive "set" directive. Different options
87          00072 ; can be specified as necessary, provided you know what they do.
88          00003FFF ConfigurationWord set      _DEBUG_OFF
89          00002FFF ConfigurationWord set      ConfigurationWord & _LVP_OFF
90          000027FF ConfigurationWord set      ConfigurationWord & _FCMEN_OFF
91          000023FF ConfigurationWord set      ConfigurationWord & _IESO_OFF
92          000020FF ConfigurationWord set      ConfigurationWord & _BOR_OFF
93          000020FF ConfigurationWord set      ConfigurationWord & _CPD_OFF
94          000020FF ConfigurationWord set      ConfigurationWord & _CP_OFF
95          000020FF ConfigurationWord set      ConfigurationWord & _MCLRE_ON
96          000020FF ConfigurationWord set      ConfigurationWord & _PWRTE_OFF
97          000020F7 ConfigurationWord set      ConfigurationWord & _WDT_OFF
98          000020F4 ConfigurationWord set      ConfigurationWord & _INTRC_OSC_NOCLKOUT
99
100         00084
101         00085 ; The '_CONFIG' directive is used to embed configuration data within
102         00086 ; a .asm file. The labels used in each "set" directive are located
103         00087 ; in the respective p16f884.inc file. See the data sheet for
104         00088 ; additional information on the meaning of the bits within the
105         00089 ; configuration word. The PIC16F884 has two configuration words,
106         00090 ; so an address must be specified in the _CONFIG directive.
107         00091 ; The first is at address 0x2007.
107    2007   20F4 _CONFIG 0x2007 ,ConfigurationWord
108
109
110    00003FFF ConfigurationWord set      _WRT_OFF
111    00003FFF ConfigurationWord set      ConfigurationWord & _BOR40V
112    2008   3FFF _CONFIG 0x2008 ,ConfigurationWord
113
114
115    00098 ;***** VARIABLE DEFINITIONS
116    00100 ; The first two variables defined specify storage locations used for
117    00101 ; context switching when interrupts occur. The locations specified are
118    00102 ; in a part of memory that is present no matter which memory bank is
119    00103 ; currently in use.

```

120  
121  
122 LOC OBJECT CODE LINE SOURCE TEXT  
123 VALUE  
124  
125 00104  
126 00105 ; The first two variables defined specify storage locations used for  
127 ; context switching when interrupts occur. The locations specified are  
128 ; in a part of memory that is present no matter which memory bank is  
129 ; currently in use.  
130 00000070 00109 w\_temp EQU 0x70 ; Variable used for saving W  
131 00000071 00110 status\_temp EQU 0x71 ; Variable used for saving STATUS  
132 00111  
133 00112 ; Program storage areas that do not need to be available in all banks  
134 ; should be specified next.  
135 00114 cblock 0x20  
136 00115 ; These registers are used by the Divide subroutine  
137 00000020 00116 DU ; 8-bit high-order extension to the dividend  
138 00000021 00117 DH,DL ; 16-bit dividend  
139 00000023 00118 QH,QL ; 16-bit quotient  
140 00000025 00119 VL ; 8-bit divisor  
141 00000026 00120 Count ; 8-bit counter for use by the Division subroutine  
142 00000027 00121 RL ; 8-bit remainder  
143 00122  
144 00123 ; These registers are used by the main program and by the  
145 ; Timer 1 interrupt routine for storing a snapshot value  
146 ; of Timer 1  
147 00000028 00126 T1H,T1L  
148 00127  
149 00128 ; These registers are used by the main program  
150 0000002A 00129 ReturnCode ; A place to store the return code from the Division  
151 ; subroutine so that it can be examined  
152 0000002B 00130 Temp ; A place for temporary storage.  
153 00132 endc  
154 00133  
155 00134 ; The Divide subroutine returns one of the following flags in the W  
156 ; register to indicate whether it succeeded or not.  
157 00000000 00136 DivideByZero equ 0  
158 FFFFFFFF 00137 SuccessfulDivision equ -D'1'  
159 00138  
160 00139  
161 00140 ; Pin allocations  
162 ; The display has four seven-segment displays in it:  
163 ; -888 (a sign followed by three digits)  
164 00143 ;  
165 00144 ; Most significant 7-segment display (Digit 2)  
166 00145 ;RA0 Segment g of the leftmost display (for a minus sign)  
167 00146 ;RA1 Segment g  
168 00147 ;RA2 Segment f  
169 00148 ;RA3 Segment e  
170 00149 ;RA4 Segment d  
171 00150 ;RA5 Segment c  
172 00151 ;RA6 Segment b  
173 00152 ;RA7 Segment a  
174 00153 ;  
175 00154 ; Next most significant 7-segment display (Digit 1)  
176 ;RB0/INT Input from the MAX6576 Temperature Sensor  
177 00156 ;RB1 Segment g

```

181 LOC OBJECT CODE      LINE SOURCE TEXT
182     VALUE

183
184         00157 ;RB2      Segment f
185         00158 ;RB3      Segment e
186         00159 ;RB4      Segment d
187         00160 ;RB5      Segment c
188         00161 ;RB6      Segment b
189         00162 ;RB7      Segment a
190         00163 ;
191         00164 ;Least significant 7-segment display (Digit 0)
192         00165 ;RC0      Segment g
193         00166 ;RC1      Segment f
194         00167 ;RC2      Segment e
195         00168 ;RC3      Segment d
196         00169 ;RC4      Segment c
197         00170 ;RC5      Segment b
198         00171 ;RC6      Segment a
199         00172 ;RC7
200         00173 ;
201         00174 ;RD0
202         00175 ;RD1
203         00176 ;RD2
204         00177 ;RD3
205         00178 ;RD4
206         00179 ;RD5
207         00180 ;RD6
208         00181 ;RD7
209         00182 ;
210         00183 ;RE0
211         00184 ;RE1
212         00185 ;RE2
213         00186 ;RE3
214         00187
215 00000007 00188 Digit0 equ PORTC
216 00000006 00189 Digit1 equ PORTB
217 00000005 00190 Digit2 equ PORTA
218 00000000 00191 SignSegment equ D'0' ; Segment g of the 7-segment display for
219                               ; the arithmetic sign of the temperature.
220 00000005 00193 SignPort   equ PORTA ; The sign bit is in PORTA.
221                               ; All seven-segment displays are common-anode types, so segments
222                               ; are illuminated when the corresponding control bit is a 0
223                               ; and they are off otherwise.
224                               00197
225                               00198 ; Port A (Digit2) initialization values
226 00000000 00199 TRISA_Init equ B'00000000' ; Bits 7 through 1 drive a 7-segment display
227 00000000 00200 ANSEL_Init equ B'00000000' ; Specify ANS7 through ANS0 for digital use.
228                               00201
229                               00202 ; Port B (Digit1) initialization values to support interrupts
230 00000001 00203 TRISB_Init equ B'00000001' ; RB0/INT is an input
231                               ; Bits 7 through 1 drive a 7-segment display
232 00000000 00205 ANSELH_Init equ B'00000000' ; Specify ANS13 through ANS8 for digital use.
233                               00206 ; These correspond to RB5 through RB0. (RB7 and RB6 have no analog use.)
234                               00207
235                               00208
236                               00209 ; Port C (Digit0) initialization values

```

```
240 LOC OBJECT CODE LINE SOURCE TEXT
241 VALUE

242
243 00000000          00210 TRISC_Init  equ B'00000000' ; Bits 7 through 1 drive a 7-segment display
244          00211
245          00212 ; Timer 1 initialization values
246          00213 ; T1GINV<7>; don't care (0)
247          00214 ;TMR1GE<6>; ignored (0)
248          00215 ; Prescale<5:4> 1:4 (10)
249          00216 ; T1OSCEN<3>; off (0)
250          00217 ; T1SYNC_N<2>; synchronized (0)
251          00218 ; TMR1CS<1>; Internal (0)
252          00219 ; TMR1ON<0>; on (1)
253 00000021          00220 T1CON_Init  equ B'00100001'
254          00221
255          00222
256          00223 ; Macros
257          00224
258          00225 ; Sets RP1 and RP0 to give the appropriate bank, specified by BankNum.
259          00226 Bankset macro BankNum
260          00227     if BankNum == 0
261          00228         bcf STATUS,RP1
262          00229         bcf STATUS,RP0
263          00230     endif
264          00231     if BankNum == 1
265          00232         bcf STATUS,RP1
266          00233         bsf STATUS,RP0
267          00234     endif
268          00235     if BankNum == 2
269          00236         bsf STATUS,RP1
270          00237         bcf STATUS,RP0
271          00238     endif
272          00239     if BankNum == 3
273          00240         bsf STATUS,RP1
274          00241         bsf STATUS,RP0
275          00242     endif
276          00243   endm
277          00244
278          00245 ; Takes a seven-segment display code stored in
279          00246 ; general purpose register CodeNumAddr,
280          00247 ; shift it left one bit,
281          00248 ; and outputs it to the specified port (Device).
282          00249 ; All of the ports have been set up to expect
283          00250 ; the display codes to be in bits 7 through 1.
284          00251 DisplayCode macro Device,CodeNumAddr
285          00252 ; Use the CodeNumAddr value and convert it
286          00253 ; to an LED display pattern
287          00254     movf   CodeNumAddr,W
288          00255     call   GetLEDPattern
289          00256 ; Save the LED pattern in a temporary location
290          00257     movwf  Temp
291          00258 ; Preserve the sign-bit's value by copying it to the C bit.
292          00259     if Device == SignPort
293          00260         btfsc  SignPort,SignSegment
294          00261         bsf    STATUS,C
295          00262         btfss  SignPort,SignSegment
```

297  
298  
299 LOC OBJECT CODE LINE SOURCE TEXT  
300 VALUE  
301  
302 00263 bcf STATUS,C  
303 00264 else  
304 00265 bcf STATUS,C ; Otherwise shift in a 0  
305 00266 endif  
306 00267 rlf Temp,W  
307 00268 movwf Device  
308 00269 endm  
309 00270  
310 00271  
311 00272 ;\*\*\*\*\*  
312 0000 0000 018A 00273 ORG 0x000 ; Processor reset vector  
313 0000 0000 00274 clrf PCLATH ; Ensure page bits are cleared  
314 00275  
315 0001 280E 00276 goto main ; Go to beginning of program  
316 00277 ; This jumps around the interrupt-servicing  
317 00278 ; code that has to be a location 0x004.  
318 00279  
319 0004 0004 00280 ORG 0x004 ; Interrupt vector location  
320 0004 0004 00281 ; Save W and STATUS in temporary locations  
321 0004 00F0 00282 movwf w\_temp ; Copy W to a temporary location  
322 0005 0E03 00283 swapf STATUS,w ; Swap STATUS into W  
323 00284 ; Swap is used because it does not change  
324 00285 ; any bits in the STATUS register.  
325 0006 00F1 00286 movwf status\_temp ; Move STATUS into a temporary location  
326 00287  
327 00288 ; ISR code can either be inserted here or be reached by a subroutine  
328 00289 ; call in the example shown here.  
329 00290  
330 00291 ; Check to see if external interrupt occurred.  
331 0007 188B 00292 btfsc INTCON,INTF  
332 0008 2089 00293 call DealWithExternalInterrupt ; If so, deal with it  
333 00294  
334 00295 ; At this point we have checked as many interrupt sources as we  
335 00296 ; care to recognize and so we can return from the interrupt.  
336 00297  
337 0009 0E71 00298 swapf status\_temp,w ; Swap status\_temp into w. This  
338 00299 ; Restores the bank to what it was before  
339 00300 ; the interrupt occurred.  
340 000A 0083 00301 movwf STATUS ; Restore pre-ISR STATUS register contents  
341 000B 0EF0 00302 swapf w\_temp,f ; The goal is to copy w\_temp into W  
342 00303 ; without altering any STATUS bits  
343 000C 0E70 00304 swapf w\_temp,w  
344 000D 0009 00305 retfie ; Return from interrupt  
345 00306  
346 000E 00307 main  
347 00308 ; Initialize ports  
348 00309 Bankset 0 ; All digits are in bank 0  
349 M if 0 == 0  
350 000E 1303 M bcf STATUS,RP1  
351 000F 1283 M bcf STATUS,RP0  
352 M endif  
353 M if 0 == 1  
354 M bcf STATUS,RP1

356  
357  
358 LOC OBJECT CODE LINE SOURCE TEXT  
359 VALUE  
360  
361 M bsf STATUS, RP0  
362 M endif  
363 M if 0 == 2  
364 M bsf STATUS, RP1  
365 M bcf STATUS, RP0  
366 M endif  
367 M if 0 == 3  
368 M bsf STATUS, RP1  
369 M bsf STATUS, RP0  
370 M endif  
371 0010 30FF 00310 movlw ~0 ; Turn off all Digit2 segments, including the sign.  
372 0011 0085 00311 movwf Digit2  
373 0012 0086 00312 movwf Digit1 ; Turn off all Digit1 segments.  
374 0013 0087 00313 movwf Digit0 ; Turn off all Digit0 segments.  
375 0014 1303 00314 Bankset 1 ; Initialize TRISA, TRISB, and TRISC in bank 1.  
376 M if 1 == 0  
377 M bcf STATUS, RP1  
378 M bcf STATUS, RP0  
379 M endif  
380 M if 1 == 1  
381 0015 1683 M bcf STATUS, RP1  
382 M bsf STATUS, RP0  
383 M endif  
384 M if 1 == 2  
385 M bsf STATUS, RP1  
386 M bcf STATUS, RP0  
387 M endif  
388 M if 1 == 3  
389 M bsf STATUS, RP1  
390 M bsf STATUS, RP0  
391 M endif  
392 0016 3000 00315 movlw TRISA\_Init  
393 Message[302]: Register in operand not in bank 0. Ensure that bank bits are correct.  
394 0017 0085 00316 movwf TRISA  
395 0018 3001 00317 movlw TRISB\_Init  
396 Message[302]: Register in operand not in bank 0. Ensure that bank bits are correct.  
397 0019 0086 00318 movwf TRISB  
398 001A 3000 00319 movlw TRISC\_Init  
399 Message[302]: Register in operand not in bank 0. Ensure that bank bits are correct.  
400 001B 0087 00320 movwf TRISC  
401 00321 Bankset 3 ; Initialize ANSELH in bank 3  
402 M if 3 == 0  
403 M bcf STATUS, RP1  
404 M bcf STATUS, RP0  
405 M endif  
406 M if 3 == 1  
407 M bcf STATUS, RP1  
408 M bsf STATUS, RP0  
409 M endif  
410 M if 3 == 2  
411 M bsf STATUS, RP1  
412 M bcf STATUS, RP0  
413 M endif

```

415
416
417 LOC OBJECT CODE      LINE SOURCE TEXT
418     VALUE

419
420             M      if 3      == 3
421 001C 1703             M      bsf STATUS,RP1
422 001D 1683             M      bsf STATUS,RP0
423             M      endif
424 001E 3000             00322  movlw  ANSEL_Init
425 Message[302]: Register in operand not in bank 0. Ensure that bank bits are correct.
426 001F 0088             00323  movwf  ANSEL
427 0020 3000             00324  movlw  ANSELH_Init
428 Message[302]: Register in operand not in bank 0. Ensure that bank bits are correct.
429 0021 0089             00325  movwf  ANSELH
430             00326  Bankset 0 ; Return to Bank 0 for normal access
431             M      if 0      == 0
432 0022 1303             M      bcf STATUS,RP1
433 0023 1283             M      bcf STATUS,RP0
434             M      endif
435             M      if 0      == 1
436             M      bcf STATUS,RP1
437             M      bsf STATUS,RP0
438             M      endif
439             M      if 0      == 2
440             M      bsf STATUS,RP1
441             M      bcf STATUS,RP0
442             M      endif
443             M      if 0      == 3
444             M      bsf STATUS,RP1
445             M      bsf STATUS,RP0
446             M      endif
447             00327
448             00328 ; Initialize Timer 1
449 0024 3021             00329  movlw  T1CON_Init
450 0025 0090             00330  movwf  T1CON
451             00331
452             00332 ; Enable interrupts
453 0026 018F             00333  clrf   TMRIH      ; Clear Timer 1 preparatory to
454 0027 018E             00334  clrf   TMRI1L    ; taking first temperature measurement.
455 0028 108B             00335  bcf    INTCON,INTF ; Clear the external interrupt flag.
456 0029 160B             00336  bsf    INTCON,INTE ; Enable the external interrupt
457 002A 178B             00337  bsf    INTCON,GIE  ; Enable global interrupts
458             00338
459             00339
460 002B                 00340  loop
461             ; Convert the period of the MAX6576 waveform to a temperature.
462             ; Assume f_osc = 10 MHz, as in the notes.
463             ; Temperature = (n / 100) - 273
464             ; Place divisor (100) in VL
465 002B 3064             00345  movlw  D'0100'
466 002C 00A5             00346  movwf  VL
467             00347 ; Place saved Timer 1 value in dividend (DH:DL)
468 002D 0828             00348  movf   T1H,W
469 002E 00A1             00349  movwf  DH
470 002F 0829             00350  movf   T1L,W
471 0030 00A2             00351  movwf  DL
472 0031 2071             00352  call   Divide

```

474  
 475  
 476 LOC OBJECT CODE LINE SOURCE TEXT  
 477 VALUE  
 478  
 479                   00353 ; If remainder >= 50, round the quotient up  
 480 0032 0827 00354 movf RL,W  
 481 0033 3C32 00355 sublw D'50'  
 482 0034 1803 00356 btfsc STATUS,C  
 483 0035 2839 00357 goto RoundDown  
 484 0036         00358 RoundUp  
 485 0036 0AA4 00359 incf QL,F  
 486 0037 1803 00360 btfsc STATUS,C  
 487 0038 0AA3 00361 incf QH,F  
 488                   00362 ; No need to check for a carry out now because the  
 489                   00363 ; dividend was already divided by 10 to get  
 490                   00364 ; the quotient.  
 491 0039         00365 RoundDown  
 492         00366  
 493                   00367 ; Now subtract 273 to complete the determination of  
 494                   00368 ; the temperature  
 495                   00369 ; D'273' = H'111'  
 496 0039 3011 00370 movlw H'11'  
 497 003A 02A4 00371 subwf QL,F  
 498                   00372 ; Check for a borrow  
 499 003B 1803 00373 btfsc STATUS,C  
 500 003C 283E 00374 goto DecrementHighOrderByte  
 501                   00375 ; A borrow occurred. Do an extra decrement  
 502                   00376 ; of the high-order byte  
 503 003D 03A3 00377 decf QH,F  
 504 003E 03A3 00378 DecrementHighOrderByte  
 505 003E 03A3 00379 decf QH,F  
 506         00380  
 507                   00381 ; Decide if the temperature is negative. If  
 508                   00382 ; so, turn on the minus sign  
 509 003F 1BA3 00383 btfsc QH,7 ; This is the sign bit  
 510 0040 2843 00384 goto HandleNegativeTemperature  
 511 0041 1005 00385 bcf SignPort,SignSegment ; Don't show minus sign  
 512 0042 284A 00386 goto HandlePositiveTemperature  
 513         00387  
 514 0043         00388 HandleNegativeTemperature  
 515 0043 1405 00389 bsf SignPort,SignSegment ; Show minus sign  
 516         00390 ; Take the absolute value of the temperature  
 517 0044 09A4 00391 comf QL,F ; Take the one's complement  
 518 0045 09A3 00392 comf QH,F  
 519 0046 3001 00393 movlw 1 ; Add 1  
 520 0047 07A4 00394 addwf QL,F  
 521 0048 1803 00395 btfsc STATUS,C; Account for a carry, if any  
 522 Message[305]: Using default destination of 1 (file).  
 523 0049 0AA3 00396 incf QH  
 524         00397  
 525 004A         00398 HandlePositiveTemperature  
 526         00399 ; Display the temperature  
 527         00400 ; Store the temperature as the dividend  
 528 004A 0823 00401 movf QH,W  
 529 004B 00A1 00402 movwf DH  
 530 004C 0824 00403 movf QL,W  
 531 004D 00A2 00404 movwf DL

534  
535 LOC OBJECT CODE LINE SOURCE TEXT  
536 VALUE  
537  
538 00405 ; To extra decimal digits , divide by 10  
539 00406 ; repeatedly.  
540 004E 300A 00407 movlw D'10'  
541 004F 00A5 00408 movwf VL  
542 00409  
543 00410 ; Get the one's digit  
544 00411  
545 0050 2071 00412 call Divide  
546 00413 ; Remainder now has the one's digit.  
547 00414 DisplayCode Digit0 ,RL  
548 M ; Use the CodeNumAddr value and convert it  
549 M ; to an LED display pattern  
550 0051 0827 M movf RL,W  
551 0052 2099 M call GetLEDPattern  
552 M ; Save the LED pattern in a temporary location  
553 0053 00AB M movwf Temp  
554 M ; Preserve the sign-bit's value by copying it to the C bit.  
555 M if Digit0 == SignPort  
556 M btfsc SignPort ,SignSegment  
557 M bsf STATUS,C  
558 M btfss SignPort ,SignSegment  
559 M bcf STATUS,C  
560 M else  
561 0054 1003 M bcf STATUS,C ; Otherwise shift in a 0  
562 M endif  
563 0055 0D2B M rlf Temp,W  
564 0056 0087 M movwf Digit0  
565 00415  
566 00416 ; Get the ten's digit. The previous quotient  
567 00417 ; is the temperature divided by 10. Make  
568 00418 ; this the dividend and repeat the division.  
569 0057 0823 00419 movf QH,W  
570 0058 00A1 00420 movwf DH  
571 0059 0824 00421 movf QL,W  
572 005A 00A2 00422 movwf DL  
573 005B 2071 00423 call Divide  
574 00424 ; Remainder now has the tens digit.  
575 00425 DisplayCode Digit1 ,RL  
576 M ; Use the CodeNumAddr value and convert it  
577 M ; to an LED display pattern  
578 005C 0827 M movf RL,W  
579 005D 2099 M call GetLEDPattern  
580 M ; Save the LED pattern in a temporary location  
581 005E 00AB M movwf Temp  
582 M ; Preserve the sign-bit's value by copying it to the C bit.  
583 M if Digit1 == SignPort  
584 M btfsc SignPort ,SignSegment  
585 M bsf STATUS,C  
586 M btfss SignPort ,SignSegment  
587 M bcf STATUS,C  
588 M else  
589 005F 1003 M bcf STATUS,C ; Otherwise shift in a 0  
590 M endif

592  
593  
594 LOC OBJECT CODE LINE SOURCE TEXT  
595 VALUE  
596  
597 0060 0D2B M rlf Temp,W  
598 0061 0086 M movwf Digit1  
599 00426  
600 00427 ; Get the hundred's digit. The previous quotient  
601 00428 ; is the temperature divided by 100. Make  
602 00429 ; this the dividend and repeat the division.  
603 0062 0823 00430 movf QH,W  
604 0063 00A1 00431 movwf DH  
605 0064 0824 00432 movf QL,W  
606 0065 00A2 00433 movwf DL  
607 0066 2071 00434 call Divide  
608 00435 ; Remainder now has the hundreds digit.  
609 00436 DisplayCode Digit2 ,RL  
610 00437 M ; Use the CodeNumAddr value and convert it  
611 00438 M ; to an LED display pattern  
612 0067 0827 M movf RL,W  
613 0068 2099 M call GetLEDPattern  
614 0069 00AB M ; Save the LED pattern in a temporary location  
615 0069 00AB M movwf Temp  
616 006A 1805 M ; Preserve the sign-bit's value by copying it to the C bit.  
617 006B 1403 M if Digit2 == SignPort  
618 006C 1C05 M btfsc SignPort ,SignSegment  
619 006D 1003 M bsf STATUS,C  
620 006E 0D2B M btfss SignPort ,SignSegment  
621 006F 0085 M bcf STATUS,C  
622 00439 M else  
623 00440 M bcf STATUS,C ; Otherwise shift in a 0  
624 00441 M endif  
625 006E 0D2B M rlf Temp,W  
626 006F 0085 M movwf Digit2  
627 00437  
628 00438  
629 0070 282B 00439 goto loop  
630 00440  
631 00441 ; Interrupt Service Routine for INT  
632 00442  
633 00443 ; Here is the subroutine to handle division  
634 0071 00444 Divide  
635 0071 08A5 00445 ; Check for division by 0  
636 0072 1903 00446 movf VL,F  
637 0073 3400 00447 btfsc STATUS,Z  
638 00448 retlw DivideByZero  
639 00449 ; Clear DU (upper-order extension to the dividend)  
640 0074 01A0 00450 clrf DU  
641 00451 ; QH:QL will be initialized by the program to 00:00 = 0  
642 0075 01A3 00452 clrf QH  
643 0076 01A4 00453 clrf QL  
644 0077 01A7 00454 ; RL will be initialized by the program to 00 = 0  
645 00455 clrf RL  
646 00456 ; Loop 16 times to do the division  
647 0078 3010 00457 movlw D'16'  
648 0079 00A6 00458 movwf Count  
649 007A 00459 DivisionLoop

651  
 652  
 653 LOC OBJECT CODE LINE SOURCE TEXT  
 654 VALUE  
 655  
 656 00460 ; Shift DU:DH:DL left one bit  
 657 007A 1003 00461 bcf STATUS,C  
 658 007B 0DA2 00462 rlf DL,F  
 659 007C 0DA1 00463 rlf DH,F  
 660 007D 0DA0 00464 rlf DU,F  
 661 00465 ; Subtract DU - VL (dividend - divisor)  
 662 007E 0825 00466 movf VL,W  
 663 007F 0220 00467 subwf DU,W  
 664 00468 ; Carry bit indicates the next quotient bit, so  
 665 00469 ; shift it into the quotient QH:QL  
 666 00470 ; First, though, if it's a 1, the trial subtraction  
 667 00471 ; succeeded, so update DU  
 668 0080 1803 00472 btfsc STATUS,C  
 669 0081 00A0 00473 movwf DU  
 670 0082 0DA4 00474 rlf QL,F  
 671 0083 0DA3 00475 rlf QH,F  
 672 00476  
 673 0084 0BA6 00477 decfsz Count,F  
 674 0085 287A 00478 goto DivisionLoop  
 675 00479  
 676 00480 ; DU now holds the remainder  
 677 0086 0820 00481 movf DU,W  
 678 0087 00A7 00482 movwf RL  
 679 00483  
 680 0088 34FF 00484 retlw SuccessfulDivision  
 681 00485  
 682 0089 DealWithExternalInterrupt  
 683 00487 ; If we get here, it's because the temperature  
 684 00488 ; sensor has completed one output cycle.  
 685 00489 ; Save current value of Timer 1  
 686 0089 080F 00490 movf TMR1H,W  
 687 008A 00A8 00491 movwf T1H  
 688 008B 080E 00492 movf TMR1L,W  
 689 008C 00A9 00493 movwf T1L  
 690 008D 080F 00494 movf TMR1H,W ; Read TMR1H again.  
 691 008E 0228 00495 subwf T1H,W ; Did it change?  
 692 008F 1903 00496 btfsc STATUS,Z ; Skip if not.  
 693 0090 2895 00497 goto GoodTimer1Readout ; Good readout  
 694 00498 ; Timer1H rolled over: read it again  
 695 0091 080E 00499 movf TMR1L,W  
 696 0092 00A9 00500 movwf T1L  
 697 0093 080F 00501 movf TMR1H,W  
 698 0094 00A8 00502 movwf T1H  
 699 0095 GoodTimer1Readout  
 700 00504 ; Zeroize Timer 1 to start a new measurement.  
 701 0095 018F 00505 clrf TMR1H  
 702 0096 018E 00506 clrf TMR1L  
 703 00507 ; Clear the interrupt flag to get ready for  
 704 00508 ; another measurement.  
 705 0097 108B 00509 bcf INTCON, INTF  
 706 0098 0008 00510 return  
 707 00511  
 708 00512 ; This is a table lookup in order to retrieve a \*

711  
 712 LOC OBJECT CODE LINE SOURCE TEXT  
 713 VALUE  
 714  
 715 00513 ; desired pattern for output to a seven-segment  
 716 00514 ; display.  
 717 00515 ;  
 718 00516 ;  
 719 00517 ;  
 720 00518 ;  
 721 00519 ;  
 722 00520 ;  
 723 00521 ;  
 724 00522 ;  
 725 00523 ;  
 726 00524 ;  
 727 00525 ; Segment Hex  
 728 00526 ; Number a b c d e f g  
 729 00527 ; 0 1 1 1 1 1 1 0 7E  
 730 00528 ; 1 0 1 1 0 0 0 0 30  
 731 00529 ; 2 1 1 0 1 1 0 1 6D  
 732 00530 ; 3 1 1 1 1 0 0 1 79  
 733 00531 ; 4 0 1 1 0 0 1 1 33  
 734 00532 ; 5 1 0 1 1 0 1 1 5B  
 735 00533 ; 6 1 0 1 1 1 1 1 5F  
 736 00534 ; 7 1 1 1 0 0 0 0 70  
 737 00535 ; 8 1 1 1 1 1 1 1 7F  
 738 00536 ; 9 1 1 1 1 0 1 1 7B  
 739 00537 ; A 1 1 1 0 1 1 1 77  
 740 00538 ; b 0 0 1 1 1 1 1 1F  
 741 00539 ; C 1 0 0 1 1 1 0 4E  
 742 00540 ; d 0 1 1 1 1 0 1 3D  
 743 00541 ; E 1 0 0 1 1 1 1 4F  
 744 00542 ; F 1 0 0 0 1 1 1 47  
 745 0099 00543 GetLEDPattern  
 746 0099 0782 00544 addwf PCL,F  
 747 009A 347E 00545 retlw H'7E' ; 7-segment pattern for 0  
 748 009B 3430 00546 retlw H'30' ; 7-segment pattern for 1  
 749 009C 346D 00547 retlw H'6D' ; 7-segment pattern for 2  
 750 009D 3479 00548 retlw H'79' ; 7-segment pattern for 3  
 751 009E 3433 00549 retlw H'33' ; 7-segment pattern for 4  
 752 009F 345B 00550 retlw H'5B' ; 7-segment pattern for 5  
 753 00A0 345F 00551 retlw H'5F' ; 7-segment pattern for 6  
 754 00A1 3470 00552 retlw H'70' ; 7-segment pattern for 7  
 755 00A2 347F 00553 retlw H'7F' ; 7-segment pattern for 8  
 756 00A3 347B 00554 retlw H'7B' ; 7-segment pattern for 9  
 757 00A4 3477 00555 retlw H'77' ; 7-segment pattern for A  
 758 00A5 341F 00556 retlw H'1F' ; 7-segment pattern for b  
 759 00A6 344E 00557 retlw H'4E' ; 7-segment pattern for C  
 760 00A7 343D 00558 retlw H'3D' ; 7-segment pattern for d  
 761 00A8 344F 00559 retlw H'4F' ; 7-segment pattern for E  
 762 00A9 3447 00560 retlw H'47' ; 7-segment pattern for F  
 763  
 764 00562 END Directive: 'end of program'