Audio Reporting Experimental Envelope Temperature Sensor

Brief Summary

This article describes construction and use of an inexpensive envelope temperature "gauge." The unit, about 6 inches square, sits on the exterior of the envelope. A digital sensor reads envelope temperature and reports the same using an audio tone with simplified Morse code. A microprocessor provides smart operation. Cost of electronic components is about $15. There is no wire connection between the sensor and the basket.

Included on the *Balloon Builders Journal* CD are files for the assembler program, and the HEX file that actually programs the microprocessor. The assembler program is also listed at the end of the PDF file you are currently reading.

Hardware Overview

The photo on page 5 displays a bottom view of my unit. My unit was assembled in a small box constructed from ¾ inch thick pinewood covered with 1/16 inch model aircraft birch plywood. The entire unit weighs 8 ounces, about 240 grams.

The white wire is the shrink-wrapped cover to the Dallas DS18S20 digital temperature sensor. The black round unit is the speaker. The on-off switch is next to the speaker.

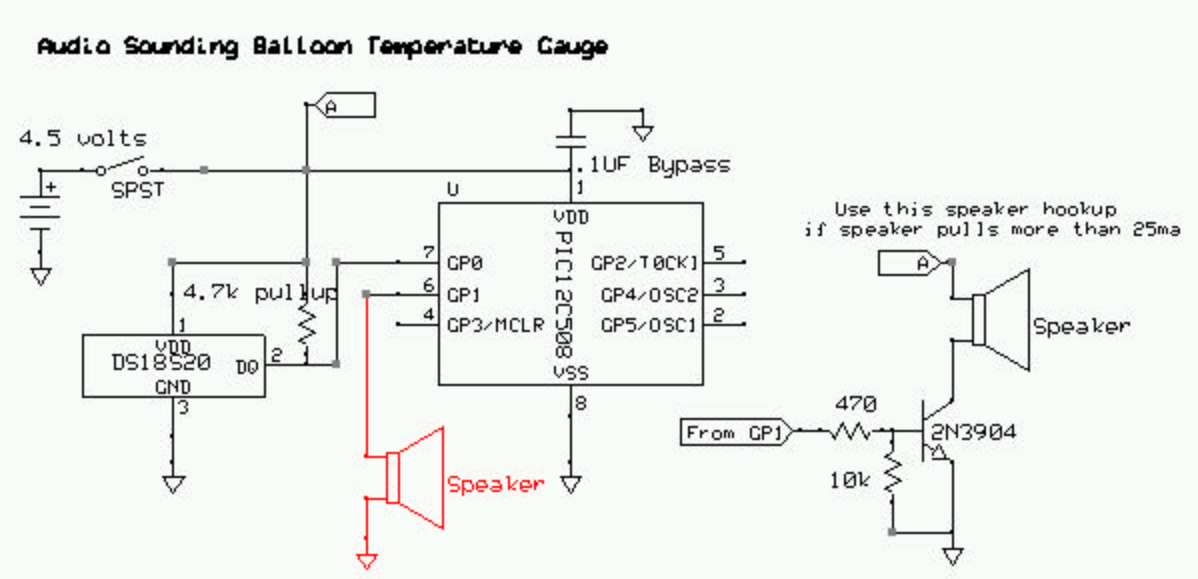
The bottom surface of the unit mounts in a pouch sewn to the envelope exterior. The sensor wire protrudes into the envelope through a small hole. The wood area around the speaker is cut out to allow flow of air, in an effort to reduce heat buildup in the speaker area.

The photo on page 6 is a top view of the unit. A "perfboard" cover has been removed from the electronics area and can just be seen at the bottom of the photo. Three AAA batteries power the unit. These remain uncovered to allow cooling flow of air. The left upper hole allows the on-off switch to be operated by a finger. The right hole is only there to permit access to the bottom of the switch.

The lower right opening contains the electronics. All the electronics are mounted on half of a Radio Shack 276-159B circuit board. The white connectors allow me to disconnect the board from the speaker and temperature sensor. The battery connects using the red and green covered homemade connectors. All connectors could be eliminated and wires directly soldered to the board. The little wood block, next to the white three-pin connector is my sophisticated effort to ensure the sensor connector doesn't wobble loose.

The perfboard cover is screwed at each of four corners over the electronics area.

Below is an electronic diagram of the sensor. Note seven components. It would be difficult to get simpler than this! However, if the speaker pulls more than 25 milliamps, replace the red speaker connection with the switching transistor arrangement shown on the right. The connector marked "A" on the speaker goes to the 4.5 volt connection. So a higher current draw requires two more resistors and one common NPN transistor.



The .1 UF bypass capacitor simply ensures the battery power remains "clean." The 4.7k-ohm pull-up resistor keeps the data pin on the DS18S20 sensor at 4.5 volts unless the PIC microprocessor or temperature sensor pulls the line to ground. The PIC and sensor communicate with each other using serial data transfer based on the Maxim "1-wire" protocol. The PIC microprocessor is a $1.00 unit that has all the smarts. The diagram shows a one-time programmable 12C508 chip. The newer 12F508 chip is a better choice for program development as it can be programmed more than once.

How It Works

When the unit is first turned on it sends a pair of audio notes every 10 seconds. This is to inform the pilot the unit is functioning.

The unit continues to send a pair of audio notes until 140 degrees F is reached in the envelope. Once this temperature is reached a five-minute timer begins. During this five minutes temperature is reported every ten seconds as passengers are loaded and the balloon is brought to equilibrium.

At the end of five minutes, temperature reporting slows to once a minute.

Anytime the temperature drops below 140 degrees, the sequence above repeats.

The 140-degree transition point is arbitrary. It assumes balloons rarely fly with a temperature below 140 degrees. However, a pilot with a protracted preflight procedure could hot inflate below 140 degrees and only exceed the transition temperature as passengers are brought onboard.

Audio Reporting

The unit has five-degree Fahrenheit resolution. Reporting is based on modified Morse code. The code is based on two tones:

A DAH is a longer, lower frequency tone.

A DIT is a shorter, higher frequency tone.

The tone pair, as the unit is first turned on, is a DAH-DIT, at ten-second intervals.

Numbers are reported using the following table:

Number 1	DIT
Number 2	DIT-DIT
Number 3	DIT-DIT-DIT
Number 4	DIT-DIT-DIT-DIT
Number 5	DIT-DIT-DIT-DIT-DIT
Number 6	DAH
Number 7	DAH-DAH
Number 8	DAH-DAH-DAH
Number 9	DAH-DAH-DAH-DAH
Number 0	DAH-DAH-DAH-DAH-DAH

The unit reports temperatures between 140 degrees and 255 degrees. Temperature is reported using two or three numbers, with a time space between each number. The first number is always a one or a two so the first number will always be a DIT or a DIT-DIT.

The second number can be a 0 through 9.

If the third number is a 0, nothing is heard, a DIT signifies a 5.

Examples:

DIT space DIT-DIT-DIT-DIT : 140 degrees—Simple, No?

DIT space DAH: 160 degrees—Again, simple, No?

DIT space DAH space DIT: 165 degrees—Same as previous, but last DIT adds 5.

DIT-DIT space DIT: 210 degrees—Again, simple, No?

DIT-DIT space DIT space DIT: 215 degrees—Same as previous, but last DIT adds 5.

DIT-DIT space DIT-DIT-DIT-DIT-DIT space DIT : 255 degrees or HIGHER

Logically, because of the temperature reporting range, anytime the second number begins with a DAH, the temperature must be below 200 degrees and at least 160 degrees.

Outstanding Issues

This is an experimental device in its first generation. Readers certainly can identify ways to further refine its function.

The principal issue with this design is heat. If I could develop an effective insulating container the unit could be wholly contained within a balloon envelope. This would require a package that

prevents the electronics from exceeding about 125 degrees F during the course of a flight. The batteries and speaker are the critical elements. The other electronic components are less temperature sensitive. The PIC is available in an extended temperature range, good to 85 degrees C.

I chose a small speaker because I didn't want the constant reporting to drive my passengers or me batty. The balloon shape acts as speaker so minimal sound power is required. The computer program could be rewritten so a piezo sounder, rather than a speaker, could be used. Doing so would eliminate the "tone modulation." The listener would hear DIT's and DAH's as a single tone of differing length. The PIC drives the speaker with square wave tones of 500 Hz and 1200Hz. With a sounder, a single tone of differing lengths would result. My small speaker pulls 20 milliamps. Don't exceed 25ma or the PIC current limits will be exceeded.

The PIC was programmed in Microchip Technologies assembler language. Included with this article is a listing of the assembler program. It is thoroughly documented so the reader might understand the program strategy. I used the PicKit1 programmer ($36), connected to a USB port on my PC. Microchip's MPLAB programming system is available for free download at Microchip.com

An alternative Reporting Scenario

The microprocessor approach allows any number of reporting scenarios. Any variation in time and temperature can be used to create a reporting method.

I haven't developed the software yet, but the following is another possible presentation: It assumes temperature should be reported if it is climbing or if it exceeds a safe threshold, like 240 degrees F.

The sensor monitors the temperature every 10 seconds.

If the new reading is 5 degrees higher than the old reading, temperature is sounded. Otherwise, the unit remains quiet.

The new reading is sounded a second time 5 seconds later in case burner noise covered the first sounding effort.

If temperature exceeds the safe threshold level, it is sounded every 10 seconds.

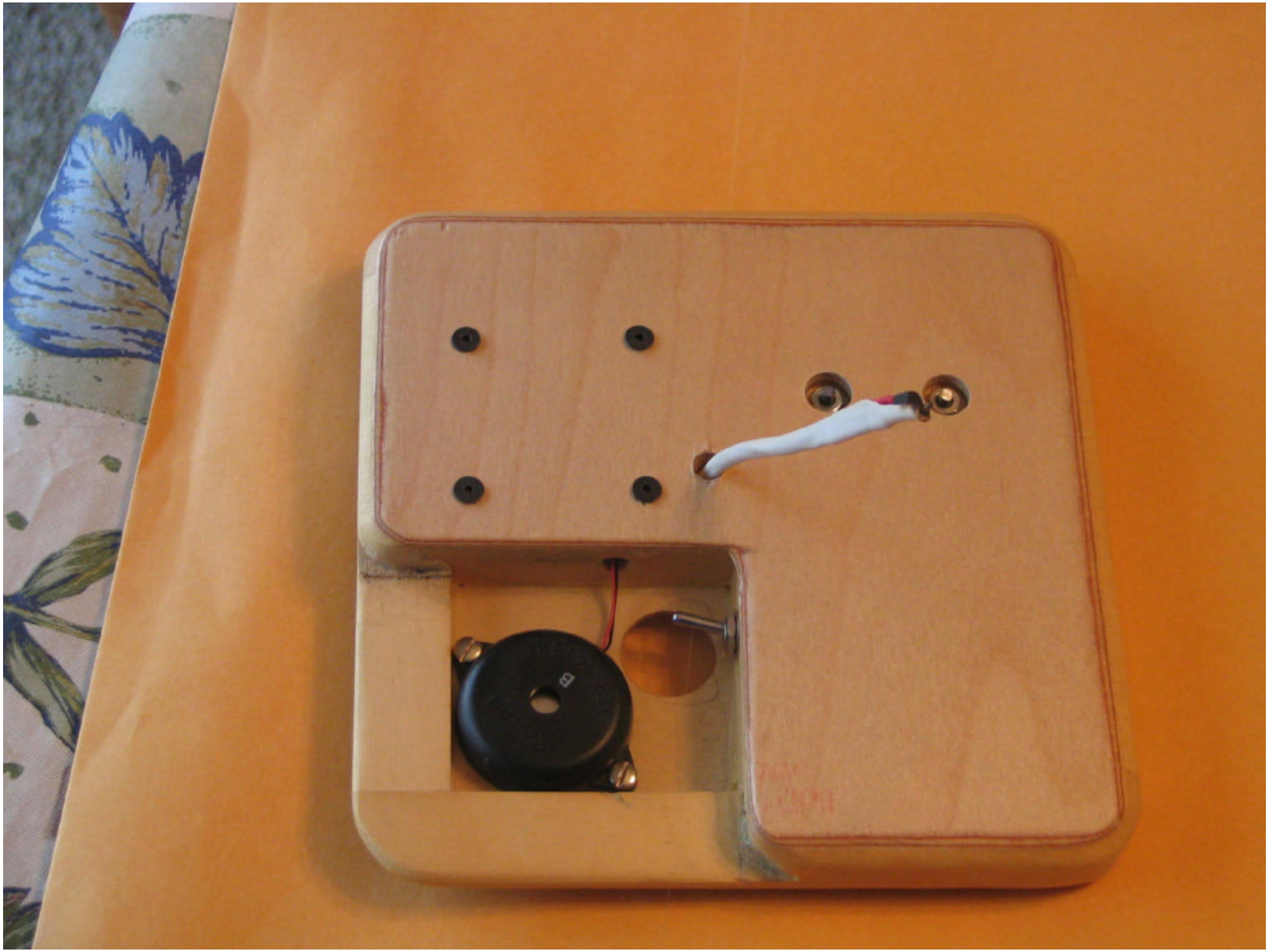
Other Possibilities

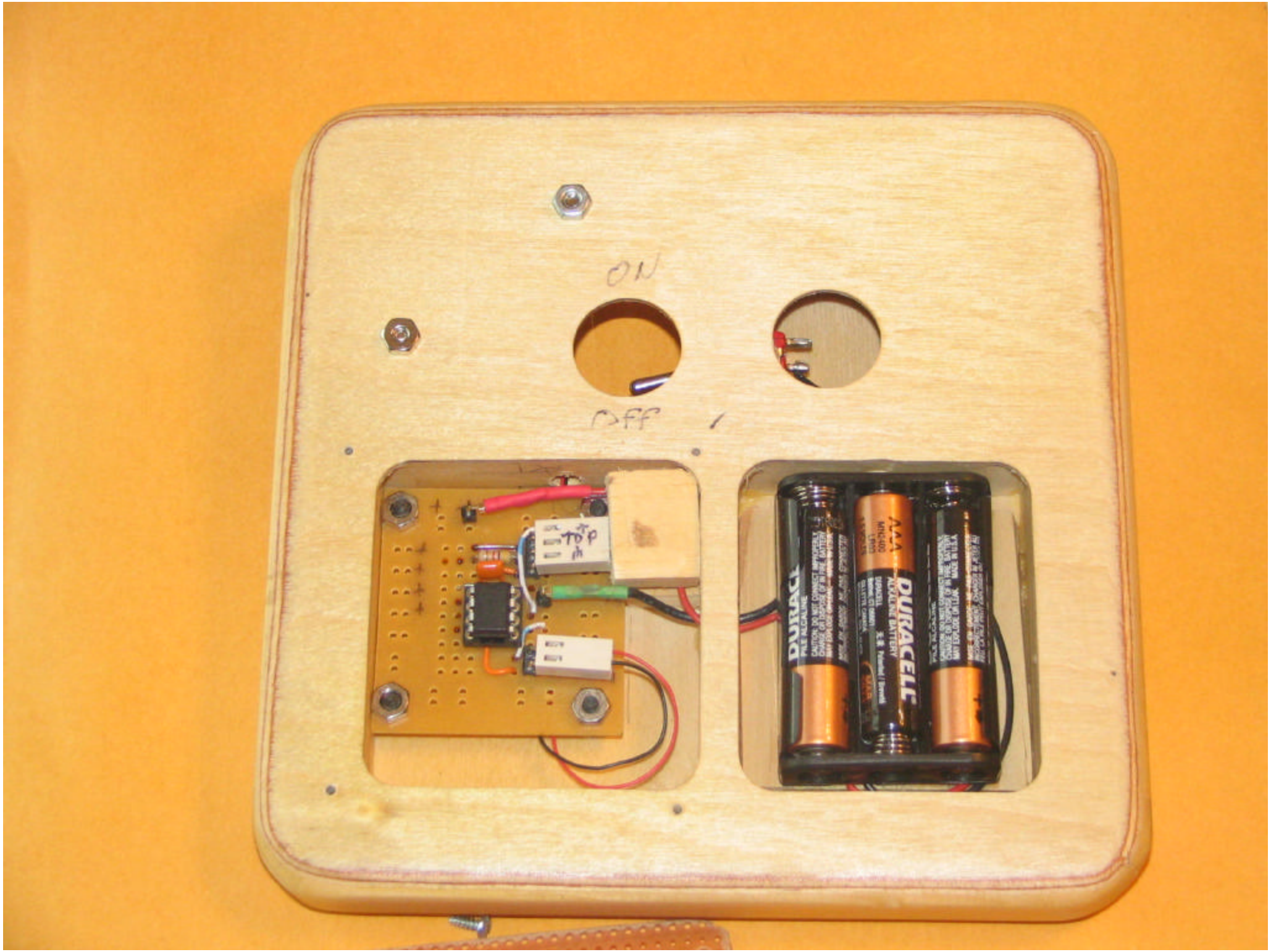
Readers might consider the following refinements:

Add a speech unit so the temperature is reported not as Morse code, but in a spoken language.

Add a microphone so the audio reporting doesn't conflict with burner noise.

Add infrared reception so a common television remote control could give the unit commands.




ON
OFF
DURACELL
DURACELL

PIC Programming Instructions

Below is a listing of the assembler file that programs the microprocessor. It is heavily commented for those who are interested in the details. The assembler file (balloontemp.asm) on the CD should be used for programming.

```
;      PROGRAM balloontemp.asm  7/6/2005 by Bob LeDoux

;This program works.
;
; Performs temperature measurement using DS1820 and 12F508.
; Data from DS1820, pin 2, communicates with 12F508 GPIO,0.
; The data line to the DS1820 must be pulled high w/ 4.7K resistor.
; The tone out to a speaker is GPIO,1.

       LIST P=12F508

;Configuration:
;      MCLR enabled
;      Code Protection off
;      Watchdog timer disabled
;      Internal RC oscillator


PCL           EQU    0x02
STATUS EQU    0x03
OSCAL         EQU    0x05
GPIO          EQU    0x06
C             EQU    0

       Constant      BAS_VAR=0x07

WARMUP        EQU    BAS_VAR+0
TEMP_F        EQU    BAS_VAR+1
TEMP_F_W              EQU    BAS_VAR+2
_N                    EQU    BAS_VAR+3
TONE_L        EQU    BAS_VAR+4
TONE_P        EQU    BAS_VAR+5
LOOP1                 EQU    BAS_VAR+6
LOOP2                 EQU    BAS_VAR+7
LOOP3                 EQU    BAS_VAR+8
O_BYTE        EQU    BAS_VAR+9
I_BYTE        EQU    BAS_VAR+9
TEMP                  EQU    BAS_VAR+10
TENS                  EQU    BAS_VAR+10

;Option: bit 7, 1=no interrupts, bit 6, 1=no pullups,
;bit 5, 0=internal clock, bit 4, edge select,
;bit 3, 1=prescaler to WDT, bits 0-2, prescaler.

;GPIO set all all output except bit 3.
;TRIS set as low except for DS1820, bit 0.

       org    0x00
       movwf  OSCAL
       movlw  b'11011111'
       option                      ;see OPTION note above
       movlw  b'00001000'
       TRIS   GPIO          ;set GPIO as output, except MCLR
       movlw  b'00001001'
       movwf  GPIO          ;set speaker, MCLR high
```

```
       movlw  .30                ;set at 30 for 5 minutes, sets
       movwf  WARMUP        ;10 second reports for 5 minutes

       goto   Begin         ;skip tables and subroutines

;TABLES-----------------------------------------------------

                            ;Sensor returns temperature in .5 degree increments.
                            ;thus value of 200 means reading of 100 degress Celcius
TEMP_TABLE          ;Converts sensor reading to F degrees
       addwf  PCL, f
       retlw  .140   ;offset 0 or 60C or 120 from sensor
       retlw  .145   ;offset 1
       retlw  .145   ;offset 2
       retlw  .150   ;offset 3
       retlw  .155   ;offset 4
       retlw  .160   ;offset 5
       retlw  .160   ;offset 6
       retlw  .165   ;offset 7
       retlw  .170   ;offset 8
       retlw  .175   ;offset 9
       retlw  .175   ;offset 10
       retlw  .180   ;offset 11
       retlw  .180   ;offset 12
       retlw  .185   ;offset 13
       retlw  .190   ;offset 14
       retlw  .195   ;offset 15
       retlw  .195   ;offset 16
       retlw  .200   ;offset 17
       retlw  .205   ;offset 18
       retlw  .205   ;offset 19
       retlw  .210   ;offset 20
       retlw  .215   ;offset 21
       retlw  .215   ;offset 22
       retlw  .220   ;offset 23
       retlw  .225   ;offset 24
       retlw  .230   ;offset 25
       retlw  .235   ;offset 26
       retlw  .235   ;offset 27
       retlw  .240   ;offset 28
       retlw  .245   ;offset 29
       retlw  .250   ;offset 30
       retlw  .250   ;offset 31
       retlw  .255   ;offset 32
       retlw  .255   ;offset 33

;tables MorseCt and MorseT translate the number of 10's into tones.
;for example, assume temperature of 160 which has 6 tens.  MorseCt
; will return the value ".10".  MorseT will then go down 10 rows
;to the last Dah before the goto command.  One Dah will be sent.

MorseCt       addwf  PCL, f ;Translates numbers into dits and dahs
       retlw  .6            ;number 0
       retlw  .4            ;number 1
       retlw  .3            ;number 2
       retlw  .2            ;number 3
       retlw  .1            ;number 4
       retlw  .0            ;number 5
       retlw  .10           ;number 6
       retlw  .9            ;number 7
       retlw  .8            ;number 8
       retlw  .7            ;number 9
```

```
MorseT                    ;sounds Dits and Dahs based on table MorseCt, above

       addwf  PCL, f
       call   Dit          ;number 5, offset 0
       call   Dit          ;number 4, offset 1
       call   Dit          ;number 3, offset 2
       call   Dit          ;number 2, offset 3
       call   Dit          ;number 1, offset 4
       goto   MorseRet
       call   Dah          ;number 0, offset 6
       call   Dah          ;number 9, offset 7
       call   Dah          ;number 8, offset 8
       call   Dah          ;number 7, offset 9
       call   Dah          ;number 6, offset 10
       goto   MorseRet

;SUBROUTINES-----------------------------------------------

; Routines for "1-Wire" serial data transfers.

INIT:                             ;initializes DS1820
       call PIN_HI
       call PIN_LO

       movlw  .70                 ;enter 50 for 500 microsecond delay
       movwf  LOOP1
       call   DELAY_10USEC

       call PIN_HI

       movlw  .50                 ;enter 50 for 500 microsecond delay
       movwf  LOOP1
       call   DELAY_10USEC
       retlw  0

IN_BYTE:                          ;returns byte in w
       movlw  .8                  ;8 bits make a byte
       movwf  _N                  ;_N is bit counter
       clrf   I_BYTE        ;this is reported temperature
IN_BYTE_1:
       call   PIN_LO
       NOP
       call   PIN_HI

       movlw  b'00001001'   ;Set pin to receive data
       tris   GPIO          ;from DS1820.
       NOP                              ;wait to allow Ds1820 to stabilize
       NOP
       NOP
       NOP
       movf   GPIO, w             ;move gpio pins to w, to read bit 0
       movwf  TEMP          ;move gpio pins to temp

       btfss  TEMP, 0             ;test the data_pin; set = 0
       bcf    STATUS, C           ;if so clear status bit c
       btfsc  TEMP, 0             ;test data-pin; clear = 1
       bsf    STATUS, C           ;if so set status bit c

       rrf    I_BYTE, f           ;rotate for next bit; carry in c
       movlw  .7                  ;ENTER 6 FOR 60 USEC DELAY
       movwf  LOOP1
       call   DELAY_10USEC
```

```
       movlw  b'00001000'   ;finished reading bit from DS1820
       TRIS   GPIO          ;allow PIC to transit to DS1820
       decfsz _N, f         ;count down counter _N
       goto   IN_BYTE_1     ;return for next bit
       retlw  0

OUT_BYTE:                           ;sends byte out the data_pin
       movlw  .8                    ;8 bits make a byte
       movwf  _N                    ;_N is bit counter

OUT_BYTE_1:
       rrf    O_BYTE, f             ;rotate the byte, picking up bits
       btfss  STATUS, C     ;test determines 0 or 1 bit out
       goto   OUT_0
       goto   OUT_1

OUT_BYTE_2:
       decfsz _N, f         ;count down counter for 8 bits
       goto   OUT_BYTE_1    ;get next bit
       retlw  0

OUT_0:                              ;sends out a 0 bit
       call   PIN_LO        ;sets data_pin low
       movlw  .7                    ;ENTER 6 FOR 60 USEC DELAY
       movwf  LOOP1
       call   DELAY_10USEC
       call   PIN_HI        ;sets data_pin high
       goto   OUT_BYTE_2

OUT_1:                              ;sends out a 1 bit
       call   PIN_LO        ;sets data_pin low
       call   PIN_HI        ;sets data_pin high
       movlw  .7                    ;ENTER 6 FOR 60 USEC DELAY
       movwf  LOOP1
       call   DELAY_10USEC
       goto   OUT_BYTE_2

PIN_HI                              ;sets data_pin high
       bsf    GPIO,0
       retlw  0

PIN_LO                              ;sets data_pin low
       bcf    GPIO,0
       retlw  0

                                        ;Dit and Dah are same code except for timing
                                        ;tone is created by setting and clearing pin.
                                        ;this creates square wave whose frequency is
                                        ;determined by time high and low.

Dit
       movlw  .30                   ;set time of each tone swing-30
       movwf  TONE_L
       movlw  .254          ;each sound 254 cycles long
       movwf  TONE_P
ToneC
       bsf    GPIO, 1                   ;tone goes out this pin
       movf   TONE_L, w
       movwf  LOOP1
       call   DELAY_10USEC
       bcf    GPIO, 1
       movf   TONE_L, w
```

```
       movwf  LOOP1
       call   DELAY_10USEC
       decfsz TONE_P, f
       goto   ToneC
       movlw  .1                  ;creates 1/4 second space after dit
       movwf  LOOP1
       call   Delay
       retlw  0

Dah
       movlw  .100          ;set time of each tone swing-100
       movwf  TONE_L
       movlw  .150          ;each sound 150 cycles long
       movwf  TONE_P

ToneC1
       bsf    GPIO, 1                    ;tone goes out this pin
       movf   TONE_L, w
       movwf  LOOP1
       call   DELAY_10USEC
       bcf    GPIO, 1
       movf   TONE_L, w
       movwf  LOOP1
       call   DELAY_10USEC
       decfsz TONE_P, f
       goto   ToneC1
       movlw  .1                  ;creates 1/4 second space after dah
       movwf  LOOP1
       call   Delay
       retlw  0

DELAY_10USEC:        ;sets delay of 10usec for each increment of loop1

       nop
       nop
       nop
       nop
       nop
       nop
       nop
       decfsz LOOP1, f
       goto   DELAY_10USEC
       retlw  0

Delay
Outer                              ;Loop1 sets 1/4 second per increment

       movlw  .250          ;make 250 for about .25 seconds
       movwf  LOOP2

Middle
       movlw  .110          ;make 110 for 1 millisecond
       movwf  LOOP3

Inner
       nop
       nop
       nop
       nop
       nop
       nop
       decfsz LOOP3,F
       goto   Inner
```

```
        decfsz LOOP2,F
        goto   Middle

        decfsz LOOP1,F
        goto   Outer

        retlw  0

;MAIN PROGRAM-----------------------------------------------

Begin
        nop
        nop
        nop
        call INIT                   ;initialize the DS1820

        movlw  0xcc          ;cch is command for 1820 to skip rom
        movwf  O_BYTE
        call   OUT_BYTE

        movlw  0x44          ;44h is command to read and store temperature
        movwf  O_BYTE
        call   OUT_BYTE

        movlw  .3                   ;time to read and store temperature
        movwf  LOOP1
        call   Delay

        call INIT                   ;initialize the DS1820

        movlw  0xcc          ;cch is command for 1820 to skip rom
        movwf  O_BYTE
        call   OUT_BYTE

        movlw  0xbe          ;beh is command to transmit out temperature
        movwf  O_BYTE
        call   OUT_BYTE

        call   IN_BYTE              ;read the temperature, only the 1st.

        call INIT                   ;initialize the DS1820

;I_BYTE now holds value of degrees Celcius times 2.

;-----------------------------------------------
                            ;This section takes the DS1820 output
                            ;which is celcius temperature times 2.
                            ;thus 100 celcius reports out as 200
                            ;binary.  That number is divided by 4.
                            ;Then 30 is subtracted from the result.
                            ;This allows a simple table to convert
                            ;celcius into F degrees, with about
                            ;5 degrees F resolution.  A converted
                            ;value of 0 represents 140F.  The table
                            ;then increments to 255F.

        movf   I_BYTE, w    ;copy into working storage

        bcf    STATUS, C            ;clear C so rrf is correct
        rrf    I_BYTE, f            ;divide the value
        bcf    STATUS, C            ;clear C so rrf is correct
        rrf    I_BYTE, f            ;by 4
```

```
       movlw  .30                ;decrement by 30 to index
       subwf  I_BYTE, w     ;table read to 0

       btfss  STATUS, C     ;C bit reports 140 degrees or less
       goto   TooLow        ;temp is below 140 degrees send dah dit
       call   TEMP_TABLE    ;temp above 140F report temp as audio

       movwf  TEMP_F
       movwf  TEMP_F_W
;-----------------------------------------------
                            ;This section breaks the F temp
                            ;down into three bytes.  The first
                            ;is the number of 100's, the second
                            ;is the number of 10's, and the
                            ;third is the remaining five's.



       clrf   TENS          ;clear counter for number of tens

       movlw  .200          ;check for no 100's
       subwf  TEMP_F_W, f
       btfss  STATUS, C     ;neg result =200+, pos =100+
       goto   OneHund              ;result was 100+
       call   Dit                  ;200 means two dits of audio
       call   Dit
       movlw  .3                   ;creates space before audio tens
       movwf  LOOP1
       call   Delay

       movlw  .200
       subwf  TEMP_F, f     ;subtract 200 for 10's calculation
       goto   TenPro

OneHund
       call   Dit                  ;100 means one dit
       movlw  .3                   ;creates space before audio tens
       movwf  LOOP1
       call   Delay

       movlw  .100
       subwf  TEMP_F, f     ;subtract 100 for 10's calculation

TenPro                             ;determine # of 10's by sucessive subtraction
       movlw  .10                  ; 10 is unit of decrement
       subwf  TEMP_F, f     ;subtract 10 from number = 0 to 99
       btfss  STATUS, C     ;negative result means too much sub
       goto   NegRes
       incf   TENS, f              ;positive result means another 10's
       goto   TenPro        ;repeat subtracting 10's
NegRes
       addwf  TEMP_F, f     ; neg result means 1 too many 10's taken
                                   ;so add back 10
       nop

;-------------------------------------------------------

TestSt              ;This section translates hundreds, tens,
                           ;and fives into morse code calls for dits
                           ;and dahs.  MorseT sends the dits and
                           ;dahs.  Relative addressing, set by
```

```
                        ;MorseCt sets the number of dits and
                        ;dahs for the numbers.  The Five
                        ;is sent as a dit for five and
                        ;no sound for zero.

                                    ;100's were sent in preceding section
       movf   TENS, w           ;Number of 10's
       call   MorseCt           ;get index for number of 10's
       nop
       goto   MorseT        ;apply index to sound send table

MorseRet
       movlw  .3                ;creates space before fives report
       movwf  LOOP1
       call   Delay

TestFive
       movlw  .5                ;Test for fives
       subwf  TEMP_F, f     ;subtract 5 from remaining number 0 through 9
       btfss  STATUS, C     ;test of 0 or 1 fives
       goto   Wait          ;no fives exist
       call   Dit               ;sound Dit because five exists
       goto   Wait          ;yes this instruction is redundant.

Wait                            ;5 minutes of 10 second reporting then 1 minute

       decfsz WARMUP, f     ;Warmup counts down for 5 minutes
       goto   DlayS         ;go to 10 second delay
       incf   WARMUP, f     ;warmup is 0, add 1 to keep from starting over
       movlw  .240          ;Count for 1 minute delay
       movwf  LOOP1
       call   Delay
       goto   Begin

DlayS
       movlw  .40               ;wait 10 seconds before next report.
       movwf  LOOP1
       call   Delay
       goto   Begin

TooLow                          ;less than 140 degrees, sound Dah-Dit
       movlw  .40               ;repeat every 10 seconds
       movwf  LOOP1
       nop
       call   Dah
       call   Dit
       movlw  .40
       movwf  LOOP1
       call   Delay
       goto   Begin

       END
```