"Jim’s Toy" was designed to be an electronic practical joke. Jim asked me to design a device that would wake up at semi-random intervals and beep for a few seconds. In addition, the device should:

- Be quiet for a fairly long period of time after initially being powered up.
- Have a very long battery life.
- Be fairly small.

The idea was to hide a dozen of these beepers in electrical outlets, inside furniture, up the chimney, etc. The long startup time was to allow Jim time to plant the device and leave the premises, hopefully alleviating Mike's suspicions. These beepers are high enough frequency that when sounding, most people will have a hard time locating the source of the beeping. The beep should repeat itself about every six hours. This way Jim is pretty sure that Mike will be home for at least one of the daily beep cycles. Jim plans to start the beepers several hours apart, so that the overall affect will be that a mysterious beeping occurs somewhere in Mike's new house in a seemingly random manner.

Because of the size and battery life requirements, I immediately thought of the Microchip PIC12C508. Using the PIC12C508, I was able to design a beeper that should last two years using a standard lithium coin cell as a power source. I added an LED and a test button to allow the operator to verify that the device is working. The test button can also be used to verify that the firmware is correct (see software description on following pages).

It may be stretching the electromechanical switch aspect of this design, but I like to think of it as an electronic equivalent of a joy buzzer. I'm sure Mike will eventually suspect Jim, and will soon thereafter call me (he knows that I'm the only one of Jim's friends capable of designing something like this). Once I tell him about the cyclic nature of the beepers he probably won't have too hard a time finding them. I also suspect I'll be designing some sort of variation for Mike to use on Jim.

**APPLICATION OPERATION**

Jim’s Toy uses a single PIC12C508. The PIC12C508 is normally in the sleep mode. It wakes on a watchdog timeout, or if the test button is pushed. The watchdog timer is normally set on its longest period (over 2 seconds). The firmware determines what caused it to wake up, either a reset (initial powerup), a watchdog timeout, or a change on one of the input lines (typically when the test button is pushed). On waking from a watchdog timeout, the firmware decrements a 16 bit counter. After initial powerup, this counter is set to a long value, to allow the beeper to sleep for a couple of days. Once the counter reaches zero, the firmware sounds a sequence of beeps. It resets the counter to a smaller value, so the beeper will now wake and make noise more frequently.

The firmware can also wake on a press of the test button. This will cause the device to make three short beeps and the LED flashes. This is intended to allow the user to test how loud the beeps are. Whenever the firmware wakes from a watchdog timeout it pulses an LED for a very short period of time. This provides a visual indication to the user that the device is functioning correctly. The LED is only on for about 10 milliseconds every 2 seconds. Because the duty cycle of the LED is so low it draw very little power. In fact because of the low duty cycle of beeping, the beeper should last for several months, perhaps as long as two years. Its current draw while sleeping is less than one microamp. A special test feature of the firmware is that it runs with a much shorter watchdog timeout period if the test button is pressed while a battery is inserted. This makes it easy to test software changes, otherwise it would take as long as two days to get through the initial count decrementing to zero. When running in special test mode, the device draws considerably more power, but operates about 128 times faster than it would in normal operation.
Microchip Tools Used:

Development Tools:
PICSTART® Plus

Assembler/Compiler version:
MPASM for Windows®, version 1.50
APPENDIX A: SOURCE CODE

; P12C508.INC Standard Header File, Version 1.01 Microchip Technology, Inc.
; NOLIST

; This header file defines configurations, registers, and other useful bits of
; information for the PIC12C508 microcontroller. These names are taken to match
; the data sheets as closely as possible.

; Note that the processor must be selected before this file is
; included. The processor may be selected the following ways:

; 1. Command line switch:
; C:\MPASM MYFILE.ASM /P12C508
; 2. LIST directive in the source file
; LIST P=12C508
; 3. Processor Type entry in the MPASM full-screen interface

; Revision History

; Rev: Date: Reason:
; 1.01 08/21/96 Removed VCLMP fuse, corrected oscillators
; 1.00 04/10/96 Initial Release

; Verify Processor

IFDEF __12C508
MESSG "Processor-header file mismatch. Verify selected processor."
ENDIF

; Register Definitions

W EQU H'0000'
F EQU H'0001'

;----- Register Files -----------------------------------------------

W EQU H'0000'
TMR0 EQU H'0001'
PCL EQU H'0002'
STATUS EQU H'0003'
FSR EQU H'0004'
OSCCAL EQU H'0005'
GPIO EQU H'0006'

;----- STATUS Bits -----------------------------------------------

PA2 EQU H'0007'
PA1 EQU H'0006'
PA0 EQU H'0005'
NOT_TO EQU H'0004'
NOT_PD EQU H'0003'
Z EQU H'0002'
Software listing:

(put here hard copy and electronic form)

;*************************************************************************

; JIMBO.ASM - Long period timer, with alarm.

; Uses watchdog timer to wake up about every 2 seconds. It keeps track of time, and after a very long time (about 36 hours), it beeps, then goes back to sleep for several hours (about 6). It also has 2 additional outputs that can be used to power external noisemakers. POUT (pin 6) goes positive when the beeper is making noise. NOUT (pin 2) is like an open collector output that sinks current when the beeper is sounding. NOUT can sink about 20 mAmp, POUT can source about 10 mAmp.

Every 10th time the watchdog wakes the processor, the LED blinks for a few milliseconds. It will be hard to see if you are not looking for it. This provides an indicator that the timer is operating correctly.

The TEST button is used for 2 tests. During normal operation, if the button is pressed and released, it will start a beep cycle. The beeper will run thru its normal sound, then the processor will go back to sleep. If the TEST button is held down while the battery is attached to the processor, the watchdog timer will run much faster than it would in normal operation. This allows for testing of the
Port usage:

- GP0 = Test button
- GP1 = POUT
- GP2 = Beeper output
- GP3 = reset
- GP4 = LED output
- GP5 = NOUT

Configuration bits:

- MCLRE = TRUE
- CP = FALSE
- WDTE = TRUE
- FOSC = INTRC

Notes:

- 1 week = 604800 seconds (a 20 bit value)
- 6 hours = 21600 seconds (a 16 bit value)
- 1 week = 28 6 hour periods

History:

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>5/31/97</td>
<td>M R Hahn</td>
</tr>
</tbody>
</table>

Tested in TEST mode (about 128 times as fast as regular mode).
Appears to work. Tests indicate that the first beep should
happen between 35 and 51 hours after powerup. Subsequent beeps
should happen at 6 to 9 hour intervals. Haven't added the POUT
and NOUT signals to the code yet.

Authors:

- Mark R. Hahn
  - 503-286-6125
  - hahndo@teleport.com

*************************************************************************

```assembly
list p=PIC12C508 ;
include "p12c508.inc";

__CONFIG _IntRC_OSC & _WDT_ON & _MCLRE_ON & _CP_OFF

__IDLOCS 1234h ;

;INDF  equ 000h ;index register
;TMRO  equ 001h ;real time clock/counter
;PCL   equ 002h ;program counter
;STATUS equ 003h ;status register
;FSR   equ 004h ;file select register
;OSCCAL equ 005h ;oscilator calibration
;GPIO  equ 006h ;IO port

;define STATUS flags
C_FLG    equ 0 ;carry
DC_FLG   equ 1 ;decimal carry
Z_FLG    equ 2 ;zero
PD_FLG   equ 3 ;power down
TO_FLG   equ 4 ;time out
RP0_FLG  equ 5 ;register page 0
RP1_FLG  equ 6 ;register page 1
GPWU_FLG equ 7 ;wake up flag
GPWUF    equ 7 ;wake up flag
```
; W equ 0
; F equ 1

; special function registers
indf equ 00h
tmr0 equ 01h
pcl equ 02h
status equ 03h
fsr equ 04h
osccal equ 05h
gpio equ 06h

;*************************************************************************
; RAM Definitions
;*************************************************************************

Tmp0 equ 007h ; a temporary location
Tmp1 equ 008h ; another temp location
Delay_cnt equ 009h

SixLo equ 00Bh ; low byte of sixes counter
SixHi equ 00Ch ; hi byte of sixes counter

Old_stat equ 00Dh
State equ 00Eh
#define sPowerup State,0 ; initial powerup state
#define sTest State,1 ; do a test beep
#define sCodeTest State,2 ; code test state
#define sAlarm State,3 ; keep track of time

Flags equ 00Fh
#define fTest Flags,0

fA equ 010h ; used by tdelay
fB equ 011h

Flash_cnt equ 012h
Beep_cnt equ 013h

SIXES_START equ d'41' ; 41 512 second periods = 6 hrs
SIXES_PWRUP equ d'250' ; 64000 2 second periods = 36 hrs

; bit assignments
c equ 0 ; carry bit
w equ 0 ; to indicate working register
z equ 2 ; zero bit

; io assignments
#define iTest gpio,0
#define oPout gpio,1
#define oBeep gpio,2
#define iReset gpio,3
#define oLED gpio,4
#define oNout gpio,5

;*************************************************************************
; RAM Definitions
;*************************************************************************

;*************************************************************************
; VECTORS:
;
; Stick some executable code in the RESET location.
;
/org 0 ;RESET vector
goto Start ;jump to Start label
;
;*************************************************************************

org 0 ;RESET vector
goto Start ;jump to Start label

;*************************************************************************

; Delay:
; Delay a bit.
;
;*************************************************************************

; Delay:
movlw 0C0h ;set up length of delay
movwf Delay_cnt ;
dellp movlw 001h ;set up length of delay
movwf Tmp0 ;
dell1 decfsz Tmp0,1 ;dec count
goto dell1 ;loop till done
clrwdt ;keep watchdog happy

decfsz Delay_cnt,1 ;dec count
goto dellp ;loop till done
retlw 000h
;
;
Big_delay
;
movlw d'100' ;
movwf Tmp1 ;
wloop call Delay ;
decfsz Tmp1,F ;
goto wloop ;
retlw 000h
;
;
;*************************************************************************

;tdelay:
; Short delay routine.
;
; Delays ((3 * 6) + 5) * W cycles.
;
; W contains a value from 0 to 255
;
;*************************************************************************

tdelay
;
movwf fB ;save count in fB
clrwdt ;keep watchdog happy
;
tdl
nop ;timing fix for 4 Mhz clock
nop
;
movlw d'6' ;loop 6 times
movwf fA ;
td2 ;
decfsz fA,1 ;dec loop counter
goto td2 ;3 * 6 cycles
;
decfsz fB,1 ;((3 * 6) + 5) * fB cycles total
goto td1 ;
;
retlw 0 ;return
;
*************************************************************************
;
Beep:
; Sound the buzzer.
;
*************************************************************************

Beep ;
movlw d'200'
movwf Beep_cnt ;

Beep_loop ;
bsf oBeep ;
movlw d'25' ;
call tdelay ;
bcf oBeep ;
movlw d'25' ;
call tdelay ;
decfsz Beep_cnt,f;
goto Beep_loop;
;
retlw 0 ;return
;
*************************************************************************
;
Flash:
; Turn on LED.
;
*************************************************************************

Flash ;
movlw d'200'
movwf Flash_cnt ;

Flash_loop ;
bcf oLED ;
call Delay ;
decfsz Flash_cnt,f;
goto Flash_loop;
;
bsf oLED ;
;
retlw 0 ;return
;
*************************************************************************
;
Start:
; Start of the program. We start way up here since we need the low
; page of memory for data tables and subroutines.
;
*************************************************************************

Start ;
movf STATUS,W;save status before it changes
movwf Old_stat ;

movlw b'00001001' ;set GP0, GP3 as inputs
tris GPIO ;
;
movlw b'00010000' ; turn off all outputs
movwf GPIO ;

btfss fTest ; check if in TEST mode
goto NoTest ; no, setup long watchdog ;
movlw '01001000'; very short watchdog timeout
option ;
;
goto PassTest;
;
NoTest ;
movlw b'01001111' ; enable pullups, assign prescaler to WDT
option ;

PassTest ;
clrwdt ; clear the watch dog
;
btfsc Old_stat,GPWUF; check for input change caused reset
goto State_check ;
;
btfss Old_stat,TO_FLG; check for a watchdog timeout
goto State_check ; if TO_FLAG = FALSE (WDT happened) check state
;
goto Powerup; go do powerup state
;
;******************************************************************************
;
; State_check:
; Figure out what state we are in, and then go to the handler
; for that state.
;
;******************************************************************************
;
State_check;
btfsc iTest ; check for test button pushed
goto Time_check ; not pushed, check time
;
btfscf Test ; check if we are in test mode
goto Time_check ; in test mode, do timekeeping
;
call Flash ; make some noise
call Beep ;
call Flash ;
call Beep ;
call Flash ;
call Beep ;

; goto Big_sleep ;

;******************************************************************************
;
; Time_check:
; Update timers, beep if it's time.
;
;******************************************************************************
;
Time_check;
decfsz SixLo, f;
goto NextTime;
;
decfsz SixHi, f;
goto NextTime;
movlw SIXES_START; 41 * 256 = 10500 2 sec periods = 6 hrs
movwf SixHi;
movlwd'0';
movwf SixLo;
   ; callBeep; make some noise
   callBig_delay;
callBeep;
callBig_delay;
callBeep;
callBig_delay;
callBeep;
callBig_delay;
callBeep;
callBig_delay;
callBeep;
callBig_delay;
callBeep;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
callBeep;
callBig_delay;
callBig_delay;
bcf oLED; turn LED on
      callDelay;
      bsf oLED; turn LED off
   gotoBig_sleep;
   ;*************************************************************************
   ; Powerup: 
   ; Flash LED, make beeper noise, check if TEST button pushed.
   ;*************************************************************************
   ; Powerup;
      callFlash; make some noise
      callBeep;
callFlash;
callBeep;
callFlash;
callBeep;
      bcf fTest;
      btfss iTest;
      bsf fTest;
      ; movlw SIXES_PWRUP; 250 * 256 = 64000 2 sec periods = 36 hrs
      movwf SixHi;
movlwd'0';
movwf SixLo;
gotoBig_sleep;
   ;*************************************************************************
   ; Big_sleep:
; Enable wake on change, and read and latch inputs. Then put
; processor to sleep.

;******************************************************************************

Big_sleep

; call Delay; delay for debouncing (make shorter later)

clrwdt

movf GPIO,W ; read and latch inputs

nop

nop

sleep ; goto sleep

nop ; these probably are not necessary

nop

goto Start ; start over

end ;