

A Better Mouse Trap

Author: Jim Nagy

Ontario, Canada

email: nagy@wwdc.com

APPLICATION OPERATION:

My application uses a PIC12C508 to produce realistic sounding mouse-like coos that all mice are sure to find seductive. The entire circuit should be imbedded in, or at least placed, near a baited mouse-trap for best effect.

The heart of the circuit is a pseudo-random number generator that determines both the time between squeaks, and the number of chirps in each squeak. In operation, the watchdog timer is used to wake the mouse up at a constant half-second rate. If the randomly determined, one to sixteen periods have passed, the mouse will emit a squeak. Squeaks consist of from one to four chirps, and each chirp is a tone that sweeps from about 5KHz to 10KHZ, in about 30mSec.

The circuit operates on two AAA dry cells, and drives a standard piezoelectric buzzer through a 4.7K resistor via a two pin push-pull output. No other components are required.

Block Diagram:

Operation is straight-forward, as described above.

Flow Chart:

Operation is straight-forward, as described above.

Graphical hardware representation:

This is probably described easier than I can draw it:

- The heart of the circuit is an 8-pin PIC12C508.
- Two AAA dry cells are connected in series to form a 3V supply, then connected with the positive lead to pin 1 of the PIC12C508, and the negative one to pin 8.

- Unused pins 2, 3, 4, and 5 are all connected to pin 1.
- Pin 7 has a 4.7K resistor connected to it with the other side of the resistor connected to either one of the wires on a piezoelectric buzzer. The other buzzer wire goes to pin 6.
- The value of the 4.7K resistor is not critical. It should be at least 1K to limit the current into the buzzer, and increased from there, to limit the volume to a pleasing level (depends on the efficiency of the buzzer).

Microchip Technology Incorporated, has been granted a nonexclusive, worldwide license to reproduce, publish and distribute all submitted materials, in either original or edited form. The author has affirmed that this work is an original, unpublished work and that he/she owns all rights to such work. All property rights, such as patents, copyrights and trademarks remain with author.

APPENDIX A: SOURCE CODE

```
MouseTrap
                                         =======
                                                                    by Jim Nagy, Sept. 1997
   A solid state mouse (the four legged kind) simulator, using the PIC12C508.
   This circuit produces realistic-sounding mouse-like coos that all mice
   are sure to find seductive. The circuit should be installed near
   a baited mouse-trap for best effect.
   This circuit is powered by a 3V source, and directly drives a
   piezoelectric buzzer. Circuit connections are as follows:
       A piezoelectric buzzer is connected through a series 4.7K
          resistor to pins 6&7 (GPO&1)
        +3V is connected to pin 1, gnd to pin 8
        pins 2,3,4, and 5 should be tied to either pin 1 or 8
    *****************
; Program equates
TMin EQU D'16'
                        ; Mouse chirps are frequency sweeps from about 5-10KHz
TMax
       EQU D'32'
                        ; the freq. is approx 166000/T
; Standard Equates
W
       EOU 0
       EOU 1
GPWUF EQU 7
PA0
              EQU 5
              EQU 4
TO
PD
              EQU
                     3
              EQU
Zero
       EQU 2
DC
              EQU 1
C
              EQU
                     0
Carry EQU
MCLRDisabled EQU 0
MCLREnabled EQU H'10'
CodeProtect
             EQU 0
NoCodeProtect EQU H'08'
WDTDisabled EQU 0
            EQU H'04'
WDTEnabled
              EQU H'02'
Int.RCOsc
              EQU H'03'
ExtRCOsc
            EQU H'01'
LPOsc
             EQU 0
; '508 Registers
INDF
      EQU H'00'
             H'01'
TMR0
        EQU
PCL
              EQU
                     H'02'
        EQU H'03'
STATUS
                     H'04'
             EQU
FSR
OSCCAL
        EQU H'05'
       EQU H'06'
; program variables
      EQU H'07'
                        ; random number variables
LByte
HByte
        EQU H'08'
                        ; numbers are generated as 2bytes+carry
CBit
        EQU H'09'
```

```
EOU H'07'
RNıım
                        ; Generated random number...same as 'LByte'
WDTimes EQU H'OA'
                        ; Mouse only chirps after 'WDTimes' wakeups
Count
        EQU H'OA'
                        ; Dual use reg - only used during a chirp
ChirpCnt EQU H'0B'
                       ; # of chirps in the squeak
CycleCnt EQU H'0C'
                       ; counts cycles during a chirp
DelayCnt EQU H'OD'
                       ; delay counter for waveform generation
   ***********
  Setting the ID words...
      ORG H'0200'
ID0
      Data.WH'0000'
      Data.WH'0000'
TD1
ID2
    Data.WH'0003'
      Data.WH'0008'
  and the Fuses...
      ORG H'OFFF'
CONFIG Data.W MCLRDisabled + NoCodeProtect + WDTEnabled + IntRCOsc
   ***********
   PIC starts here on power up...
      ORG H'00'
                   ; store the factory osc. calibration value
   subroutines must be in the low page, so jump to higher memory...
      BTFSCSTATUS,TO; check if we're here from WDT timeout
                ; no, do a full reset
      BTFSCSTATUS, PD; was a timeout, but was it a wakeup call
      GOTOInit ; no - it was a code error
      GOTOMain
                   ; yes, was a wakeup
   *************
      Chirp
   Each mouse squeak consists of a series of 1 to 4 chirps.
   Each chirp lasts about 30mS, and consists of 12 cycles at each
   frequency from a min set by TMax, to the maximum freq, set by TMin:
ChirpMOVLWTMax
                    ; get the initial waveform period
                      ; and save it
      MOVWF Count
ch1
      MOVLW D'12'
                        ; 12 cycles at each frequency
      MOVWF CycleCnt
ch2
      MOVF
            Count, W
                          ; load the count(delay) value
                       ; and produce one cycle
      BSF
             GPIO.0
      CALL
             DelayLoop
       BCF
             GPIO,0
       MOVF
             Count, W
       BSF
             GPIO,1
       CALL
             DelayLoop
       BCF
             GPIO,1
       DECFSZ CycleCnt,F ; keep repeating
       GOTO
             ch2
      DECF
             Count, F
                           ; reduce count to increase frequency
       MOVLW TMin
       SUBWF Count, W
                           ; compare to the min period value
       BTFSC STATUS, Carry; C is clear if Count<TMin
```

```
COTO
               ch1
       RETLW
       DelayLoop
   A simple delay routine...
       DECFSZ DelayCnt; save the count value
DECFSZ DelayCnt,F; count down
DelayLoop
d1
       GOTO d1
RETLW 0
                                    ; and loop,
                                     ; 'til we're done
   ************
       Random
   Generates a 'random' byte in RNum.
   Maintains a 2 byte shift register (LByte and HByte) that has an input
   provided by the XNOR of the inverse of the 13th bit and the carry out
   bit. Generates one bit at a time, so calls itself 8 times to form a byte.
Random MOVFHByte,F
                            ; have to catch the special case where all
           BTFSS STATUS, Zero ; 16 bits are 0
           GOTO
                   r1
           MOVF
                   LByte,F
           BTFSS STATUS, Zero
           GOTO
                 r1
                   TMRO,W ; both bytes are zero, seed with the low byte LByte ; with the times century.
           MOVF
           MOVWF
           BTFSS
                    STATUS, Zero ; but even the timer might read zero
           GOTO
                  LByte,F ; so then, just seed with FF
           DECE
r1
           CALL RLoop
                               ; 7 calls and a fall-through gives 8 calls...
           CALL RLoop
           CALL RLoop
           CALL RLoop
           CALL RLoop
           CALL RLoop
           CALL RLoop
RLoopMOVF CBit,F
                                    ; the XNOR is based on the carry and 13th bits
           BTFSS STATUS, Zero
                                       ; check the 'carry bit'
           GOTO CarryWas1
CarryWas0
           BTFSC HByte,4
                                            ; C=0, so check bit 13
           INCF
                  CBit,F
                                        ; if it's 1, we'll rotate in a 1
           GOTO
                  SetCarry
CarryWas1
           CLRF
                                        ; assume the new carry will be 0
           BTFSS HByte,4
                                           ; which it will be if bit13 is 1
           INCF CBit,F
                                        ; else set CBit to 1 (b13=0)
SetCarry
           CLRW
                                        ; start with W=0
           ADDWF
                 CBit,F
                                        ; adding 0 to anything forces C=0
           BTFSS STATUS, Zero
                                        ; if CBit=0, go on
           SUBWF CBit,F
                                        ; else, set C=1
                                        ; rotate the new bit into the shift reg
RotateRLF LByte,F
           RLF
                  HByte,F
           CLRF
                 CBit
                                        ; then set CBit to the current value of C
```

```
BTFSC STATUS, Carry
             INCF
                    CBit,F
             RETLW
       ***********
       Provides a 50mS delay - careful it uses Count reg!
      MOVLW D'65'
Wait
             MOVWF Count
                                     ; loop counter
             MOVLW H'FF'
             CALL
                    DelayLoop
                                     ; delay 0.77mS
             DECFSZ Count,F
                                     ; and repeat
             GOTO s1
             RETLW 0
   **********
   Power On jumps to here...either Init, or Main
   **********
Init
      CLRF
             WDTimes
                                     ; force a single chirp this time
                   ChirpCnt
             CLRF
             INCF
                   ChirpCnt,F
Main
      CLRF
             GPIO
                                     ; Init the port - WDT always clears it
             MOVLW B'00111100'
                                        ; GPO and GP1 are outputs, others are inputs
             TRIS
                    GPTO
                                         ; Set up the timers...
             CLRWDT
             MOVLW B'11001101'
                                     ; int clock to TMRO, WDT uses /32 (0.5s wakeup)
             OPTION
                                         ; no pullups, and no wakeup on change
             MOVF
                    WDTimes.F
                                     ; check if WD has timed out enough times
             BTFSC STATUS, Zero
                                     ; counted down to zero - ready for a squeak
                    Squeak
             DECF
                    WDTimes,F
                                     ; else count this time,
             SLEEP
                                        ; and wait
                                     ; A squeak is chirpcnt chirps
             CALL
                    Wait
             Chirp
Squeak CALL
                               ; with pauses in between
             DECFSZ ChirpCnt,F
             GOTO
             CLRWDT
                                     ; been busy... make sure we won't be interrupted
             CALL
                    Random
                                     ; let's get another random byte
             MOVF
                    RNum,W
                                     ; and determine the next ChirpCount...
             ANDLW
                    B'00000011'
                                     ; only use the last 2 bits for the count
             MOVWF
                    ChirpCnt
                                     ; but we can't have zero squeaks,
             INCF
                    ChirpCnt,F
                                     ; so add 1
             SWAPF
                    RNum, W
                                     ; now calculate the wakeup delay...
             ANDLW
                   B'00001111'
                                     ; only use the last 4 bits (0-8 sec delay)
             MOVWF
                    WDTimes
             SLEEP
                                     ; that's all folks
```

END

| Consumer | Appliance, | Widget, | Gadget |
|----------|------------|---------|--------|
| NOTES: | | | |