APPLICATION OPERATION:

Overview
Most of us who have read the advertisements in comic books or visited novelty stores know of the classic piece of practical joke hardware known as the "whoopie cushion". This low-tech device is essentially a rubber bag with an opening on one end. This opening is like a valve in that it allows for an easy flow of air into the bag but restricts the flow of air out of the bag. The bag is placed either on top of a seat or in-between the cushion of a sofa or easy chair. When a person sits down in the chair, air is forced out of the bag through the opening, causing the bag to emit an embarrassing noise. This results in embarrassment for the victim, and more importantly, laughter among the non-victims. With the possible exception of the victim, this laughter improves the quality of life of all concerned.

While the whoopie cushion is considered a classic by the practical jokers of the world, it possesses several flaws, which are:

1: It is not inconspicuous. If left on the top of a chair, the would-be victim can generally recognize it as something that should not be there, and he/she immediately suspects foul play. Moreover, if it is placed underneath a sofa cushion, its inflated size makes it clearly visible, and the would-be victim immediately suspects foul play.

2: It possesses a repertoire of only one sound.

3: It is not self-retriggering. It must be removed from the chair and re-inflated before it is ready for the next victim.

4: It can only be triggered when someone sits in the chair. It does not possess the capability of triggering when the victim leaves the chair.

As with many things, we look to science and technology to solve our problems. While many of the parts required to produce an electronic version this classic gag device have existed for several years, only recently with the advent of low cost, small microcontrollers such as the PIC12C508 can the dream of a high tech version of the whoopie cushion be realized. Thus, I present to the world the first (to the best of my knowledge) electronic version of the whoopie cushion. It utilizes a thin piece of piezoelectric film to sense when someone sits down in the chair or gets up from the chair, and utilizes a single chip sound record/playback device to play back one of four different pre-recorded "sounds". Overall device control is provided by a PIC12C508 8 pin microcontroller.

Hardware
The Electronic Whoopie Cushion consists of a PIC12C508 microcontroller, an AMP04 instrumentation amplifier, an LM2903 comparator, an ISD1000A single chip voice record/playback device, an LM386N-3 audio amplifier, and an LP2951 low power voltage regulator. Connections to the piezoelectric sensor, a small loudspeaker, and a 9V alkaline battery are provided using header connectors. A 2.875" by .625" piece of AMP piezoelectric film is used as a “person sitting down or getting up from the chair” sensor. The piezoelectric film will generate a voltage output when a change in stress occurs. It is deployed by either inserting it into the seat cushion or by taping it to the bottom of the seat cushion. Thus, when someone either sits down in the chair or gets up from the chair, a change in stress will occur in the sensor, and it will generate an output voltage. Because the sensor has a high output impedance and is susceptible to noise, an AMP04 instrumentation amplifier is used to signal condition the sensor output. The sensor is connected to the AMP04 as a differential signal using twisted pair wires, thus providing protection against noise via common-mode rejection. A 2.5V common mode voltage is created on each of the differential inputs to the AMP04 via a voltage divider circuit using 2M ohm resistors. Since the sensor signal is AC in nature and we are running on a single +5V supply, the output of the AMP04 must be biased to some DC level in order to work properly. This is accomplished by a voltage

Author: Michael Kirkhart
Farmington Hills, Michigan
email: kirkhart@rust.net
divider circuit on the AMP04’s reference input, which causes the AMP04 output voltage to be 2.0V with no input signal present. The output of the AMP04 goes through a low pass filter to remove any remaining noise, and is then applied to the inverting input of the comparator. The comparator output, which is normally high, is connected to the GP4 pin of the PIC12C508. Whenever our conditioned sensor signal goes above the upper trigger point of the comparator (which is set to approximately 2.6V), the comparator output goes low. When the PIC12C508 sees the comparator output go low, it randomly selects one of four pre-recorded “sounds” in the ISD1000A chip and initiates a playback cycle. The sound output of the ISD1000A is amplified by the LM386N-3 and is heard via the loudspeaker. All of this fun stuff is powered by a 9V alkaline battery, and the +5V supply needed to run the signal conditioning, the sound playback chip. The PIC12C508 is generated by an LM2951 low power voltage regulator.

For the purpose of testing, the prototype circuit was constructed with sockets for both a PIC12C508 and a PIC16C54. The software was initially written for a PIC16C54, as a PICMASTER™ emulator with a PIC16C5X pod was available for debugging. After debugging the software with the PICMASTER, it was converted for use with a PIC12C508 and re-assembled. This required only very minor changes (adding the oscillator trim instructions and changing the port pin equates), which demonstrates how easy it is to migrate designs from one variety of PICmicro™ microcontroller to another.

Software
The software consists of an initialization block, a main loop, and subroutines for:

• Delay routines

Initialization block
Invoked on a CPU reset, this code trims the on-board RC oscillator, sets up the OPTION register, and initializes the GPIO register as well as the TRIS register.

Main loop
The main loop is nothing more than a sequential state machine. In the first state, the PIC12C508 is watching the output of the comparator, waiting for it to go low. Once this occurs, we enter the second state, where the PIC12C508 waits for 20 milliseconds and then checks to see if the comparator output is still low. If it is high, then we go back to the first state; otherwise, the Electronic Whoopie Cushion is now triggered and we enter the third state. The PIC12C508 takes bits 1 and 2 of the RTCC register contents and applies them to the A5 and A6 inputs of the ISD1000A sound chip.

This randomly selects one of the four pre-recorded “sounds”. Next, it sets the ISD1000A power down input low to bring the ISD1000A out of low-power standby mode. The PIC12C508 waits for 50 milliseconds and then sets the ISD1000A chip enable input low. This initiates playback of the selected “sound”. We now enter the fourth state where the PIC12C508 waits for the ISD1000A to bring its EOM output low, which indicates the playback cycle is complete. The PIC12C508 then sets the ISD1000A chip enable and power down inputs high, thus putting the ISD1000A back into low-power standby mode. We now enter the fifth state, where the PIC12C508 looks at the output of the comparator and waits for it to go high. Once this occurs, we enter the sixth state, where the PIC12C508 waits for 1 second and then checks to see if the comparator output is still high. If it is low, we go back to the fifth state; otherwise, we enter the seventh and final state, where the PIC12C508 waits for 10 seconds before going back to the first state. At this point, the Electronic Whoopie Cushion is re-armed and ready to be triggered again.

Subroutines
• delay, L_delay - these subroutines are used to generate the various time delays needed. The delay subroutine, which was written by Philip Doucet and published in an Electronics Design magazine “Software Ideas for Design” section, generates a programmable delay. The delay duration, which is specified in instruction cycles, is passed in the delay.h and delay.c file registers. The L_delay subroutine, is used to generate delays in multiples of 10 milliseconds. For this routine, the length of the delay, in units of 10 milliseconds, is passed in the w register.

Afterthoughts
Although this design works reasonably well, there is still room for improvement. Because the inspiration for this design did not come to me until eleven days before it was due for submission, there was not enough time to produce a design as optimized for cost and functionality as I would have liked. The following are some of the possible modifications to the design to lower the cost and/or improve the functionality:

• Replace the PIC12C508 with a PIC12C671. Because the PIC12C671 is an 8 pin microcontroller with an on-board 8 bit A/D converter, the signal conditioned piezoelectric sensor output could be applied directly as an analog input. This would eliminate the need for the comparator and allow for more sophisticated trigger detection schemes using simple “signature analysis” techniques. This could reduce the possibility of false triggering due to the victim moving around in the seat as opposed to sit-
ting down or standing up. It could also allow for differentiating between sitting down and standing up conditions.

• Use a lower cost/lower capacity sound chip. The ISD1000A part was used for this design because it is readily available at any Radio Shack store. ISD makes other lower priced, lower capacity sound chips that could be used instead of the ISD1000A.

• Do we really need an instrumentation amplifier? Because the signal conditioning for the sensor required both a high impedance input and high noise immunity, I chose to use a single chip instrumentation amplifier. The gain requirement for the instrumentation amplifier, however, was fairly low; for this design, a gain of 10 was sufficient. It might be possible to use a lower cost op-amp configured as a differential amplifier to do this task. Although this would not have as high an input impedance nor as good a common mode rejection as the instrumentation amplifier, it might be good enough for this application.

• Go surface mount. If I were to go into production with this design, I would definitely make it a surface mount design. This would make the size of the circuit board smaller and would likely reduce the overall cost as well.

Flow Chart:

```
Flow Chart:

S1
comparator output = 0
Wait 20 ms
comparator output = 1
Reset
S2
comparator output = 0
Wait 50 ms
SD 1000A PD = 0
S3
comparator output = 1
Wait 50 ms
ISD 1000A EOM = 1
S4
ISD 1000A EOM = 0
comparator output = 0
S5
comparator output = 1
Wait 1 second
comparator output = 1
S6
comparator output = 0
Wait 10 seconds
comparator output = 1
S7
```
APPENDIX A: SOURCE CODE:

;************************************************
;* Program for the PIC controlled *
;* Electronic Whoopie Cushion *
;* PIC12C508 Version 1.0 written *
;* 9/20/1997 by Michael Kirkhart *
;* Version 1.01 modified 10/7/1997 *
;* by Michael Kirkhart *
;* - set configuration fuses to correct *
;* values (Internal RC oscillator, no *
;* code protect, watchdog on, internal *
;* MCLR) *
;* - corrected TRIS register setting *
;* (GP3 and GP4 are now correctly set as *
;* inputs)
;************************************************
list p=12C508 ;specifies 12C508 microcontroller
list r=DEC ;specifies decimal radix as default
list x=ON ;specifies to expand macros in listing
errorlevel 1 ;print warnings and errors only in list file

;************************
;* General system info  *
;************************

;Instruction clock frequency = 4.000 MHz
;Non-branching instruction execution time = 1 microsecond
;Fuse settings: Watchdog timer = ON
; Code Protect = OFF
; Oscillator = INTRC
; MCLR on GP3 disabled
__config 0xfee

;************************
;* CPU Register equates *
;************************

INDO equ 00 ;indirect file register
RTCC equ 01 ;real time clock/counter
PC equ 02 ;program counter
STATUS equ 03 ;status register
FSR equ 04 ;file select register (pointer)
OSCCAL equ 05 ;internal oscillator fine trim register
GPIO equ 06 ;general purpose I/O register

;************************
;* Status register bit definitions *
;************************

CARRY equ 0 ;carry/borrow flag
DCARRY equ 1 ;BCD carry/borrow flag
ZERO equ 2 ;zero flag
PDOWN equ 3 ;powerdown flag
TIMEOUT equ 4 ;watchdog timeout flag

;************************
;* GPIO bit definitions *
;************************

SNDSEL0 equ 0 ;ISD1000A A5 line (O)
SNDSEL1 equ 1 ;ISD1000A A6 line (O)
PLAY equ 2 ;ISD1000A Chip Enable line (O) (active low)
EOM equ 3 ;ISD1000A End Of Message output (I) (active low)
SITDOWN equ 4 ;output from person sitting down detector (I)(active low)
PWRDWN equ 5 ;ISD1000A Power Down line (O)

;******************************************************************************
;* Equates for register files (variables)*
;******************************************************************************

cblock 0x07

delay_l ;register file for delay value LSB
delay_h ;register file for delay value MSB
dly_tmp ;temp value for delay routine
eye ;used for loop counting in L_delay routine
jay ;used for loop counting

delay constants for 1 millisecond delay using delay routine
ONEMS_H equ 0x03 ;
ONEMS_L equ 0xe8 ;
delay constants for 10 millisecond delay using delay routine
TENMS_H equ 0x27 ;
TENMS_L equ 0x10 ;

;******************************************************************************
;* Miscelaneous equates (constants) *
;******************************************************************************

;Port A, B initialization values
GPINIT equ 00100100b ;GPIO initialization value
;GP0 and GP1 low, GP2 and GP5 high
GPTRIS equ 00011000b ;GPIO initialization value
;GP0 and GP1 and GP2 and GP5 are outputs,
;GP3 and GP4 are inputs

RETRIG equ 10 ;whoopie cushion retrigger delay value
; in seconds

delay constants for 1 millisecond delay using delay routine
ONEMS_H equ 0x03 ;
ONEMS_L equ 0xe8 ;
delay constants for 10 millisecond delay using delay routine
TENMS_H equ 0x27 ;
TENMS_L equ 0x10 ;

;******************************************************************************
;* Macro definitions *
;******************************************************************************

CLC macro ;this macro will clear the C flag
bcf STATUS,CARRY
endm

SEC macro ;this macro will set the C flag
bsf STATUS,CARRY
endm

SCC macro ;used after an instruction that affects the C flag, this macro will skip the next instruction if the C flag is clear
btfsc STATUS,CARRY
endm

SCS macro ;used after an instruction that affects the C flag, this macro will skip the next instruction if the C flag is set
btfss STATUS,CARRY
endm

SLT macro ;used after a subtract instruction, this macro will skip the next instruction if the result of the subtraction is < 0
btfsc STATUS,CARRY
endm

SGE macro ;used after a subtract instruction, this macro
Consumer Appliance, Widget, Gadget

```assembly
btfss STATUS,CARRY ; will skip the next instruction if the result endm ; of the subtraction is >= 0

SEQ macro ;used after an instruction that affects the Z
btfss STATUS,ZERO ; flag, this macro will skip the next endm ; instruction if a result is zero

SNE macro ;used after an instruction that affects the Z
btfsc STATUS,ZERO ; flag, this macro will skip the next endm ; instruction if a result is non-zero

;************************
;* Start of program     *
;************************

; actual reset vector - instruction at address 0x1ff was movlw XX, where
; XX is the calibration value to be copied into the OSCCAL register

org 0 ;start of program memory
movwf OSCCAL ;calibrate on-chip oscillator
goto start ;jump to start of program

;********************************
;* Subroutines    *
;* These must be located in the *
;* lower 256 words of program   *
;* memory                  *
;********************************

;************************************************
;* Routine to generate a time delay in multiples of 10 milliseconds from 1 ms to 2.55s      *
;* Input: W = delay length in tens of tens of milliseconds* *
;* Output: W = 0 *
;* Calls: delay *
;* Uses: delay_h, delay_l, eye, W *
;************************************************

L_delay
    movwf eye ;set loop count
Ldloop  clrwdt ;clear the watchdog timer
        movlw TENMS_L ;set
        movwf delay_l ; delay
        movlw TENMS_H ; constants for
        movwf delay_h ; 10 millisecond delay
call delay ;call delay routine
decfsz eye ;have we gone thru the loop 200 times?
goto Ldloop ;if not, do it again!
    retlw 0 ;return from subroutine

;************************************************
;* Routine for generating a programmable delay *
;* (routine written by Philip Doucet - obtained from Electronics Design - August 8, 1994, *
;* page 26ES)                             *
;************************************************

delay
    movlw 0x14 ;subtract minimum # of instructions to
    subwf delay_l ; execute this routine from requested delay
```
btfss STATUS,CARRY; check for borrow
    decf delay_h; and decrement high byte if there was one
    bcf STATUS,CARRY; divide by 4
    rrf delay_l; to determine how many times to
    bcf STATUS,CARRY; execute
    rrf delay_l; delay_l loop
    movf delay_h; check to see if
    btfsc STATUS.ZERO; delay_h = 0 and
    goto dly_30; skip delay_h loop if it is
    nop; nop equalizes timing between paths

; delay_h setup and loop

    dly_10 movlw 0x3e; since each delay_h loop needs 256 cycle, or
    movwf dly_tmp; 40h times thru inner loop of cycles, minus
    nop; cycle setup, so 40h - 2 = 3eh
    goto dly_20; add a 2 cycle delay

    dly_20 nop; inner
    decfsz dly_tmp; loop
    goto dly_20; for
    nop; delay_h
    decfsz delay_h; outer loop
    goto dly_30; for
    nop; delay_h

; delay_l setup and loop

    dly_30 movf delay_l; if delay_l
    btfsc STATUS,ZERO; = 0,
    goto dly_end; skip loop
    nop;

    dly_40 nop; loop for
    decfsz delay_l; delay_l
    goto dly_40; for
    nop;

    dly_end retlw 0; return from subroutine

;********************
;* Start of program *
;********************

    start movlw 0x0f; assign prescaler to WDT, set prescaler to
    option; 128, use internal clock for RTCC
    movlw GPINIT; initialize
    movwf GPIO; GPIO
    movlw GPTRIS; register and GPIO
    tris GPIO; TRIS register
    movlw 2; wait for 20 milliseconds for
    call L_delay; power to stabilize

    infLoop
    clrwdt; pet the dog!
    btfsc GPIO,SITDOWN; have we detected a person sitting down in the chair?
    goto infLoop; if not, check again
    movlw 2; wait for
    call L_delay; 20 milliseconds
    btfsc GPIO,SITDOWN; is sitdown detector output still active?
    goto infLoop; if not, start all over again
    rrf RTCC,w; randomly select which
    iorlw 0xfc; of the 4 sounds to play using bits 1 and 2
    movwf GPIO; of the current RTCC contents (yet keep RA2 and RA3 high)
    bcf GPIO,PWRDWN; bring ISD1000A sound chip out of standby mode
    movlw 5; wait for 50 milliseconds
    call L_delay; before activating chip enable
```assembly
bcf GPIO,PLAY ; activate ISD1000A chip enable

waitEOM
clrwdt ; we might be here awhile - better pet the dog!
btfsc GPIO,EOM ; has ISD1000A EOM signal gone low yet?
goto waitEOM ; if not, check again

bsf GPIO,PWRDWN ; put ISD1000A into standby mode

noSit
clrwdt ; we might be here awhile - better pet the dog!
btfss GPIO,SITDOWN ; is the sitdown detector still active?
goto noSit ; if so, wait until it becomes inactive
movlw 100 ; wait for
call L_delay ; 1 second
btfss GPIO,SITDOWN ; is the sitdown detector still inactive?
goto noSit ; if not, let's wait until it is!
movlw RETRIG ; initialize retrigger
movwf jay ; count value

reTrigDelay
movlw 100 ; delay
call L_delay ; for 1 second
decfsz jay ; decrement loop counter
goto reTrigDelay ; if loop counter not yet zero, delay for another second
goto infLoop ; do the whole thing over again

;***************
;* Reset vector *
;***************
; For 12C508, this location contains movlw XX, where XX is the calibration value
; for the on-board oscillator - thus the real reset vector is at address 0
org 0x1ff ; location of "reset" vector

;**************
;* End of program *
;**************
end
```