



Electromechanical Switch Replacement

Jim's Toy

*Author: Mark R. Hahn
Hahn Associates
Portland, OR
email: hahndo@teleport.com*

"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.

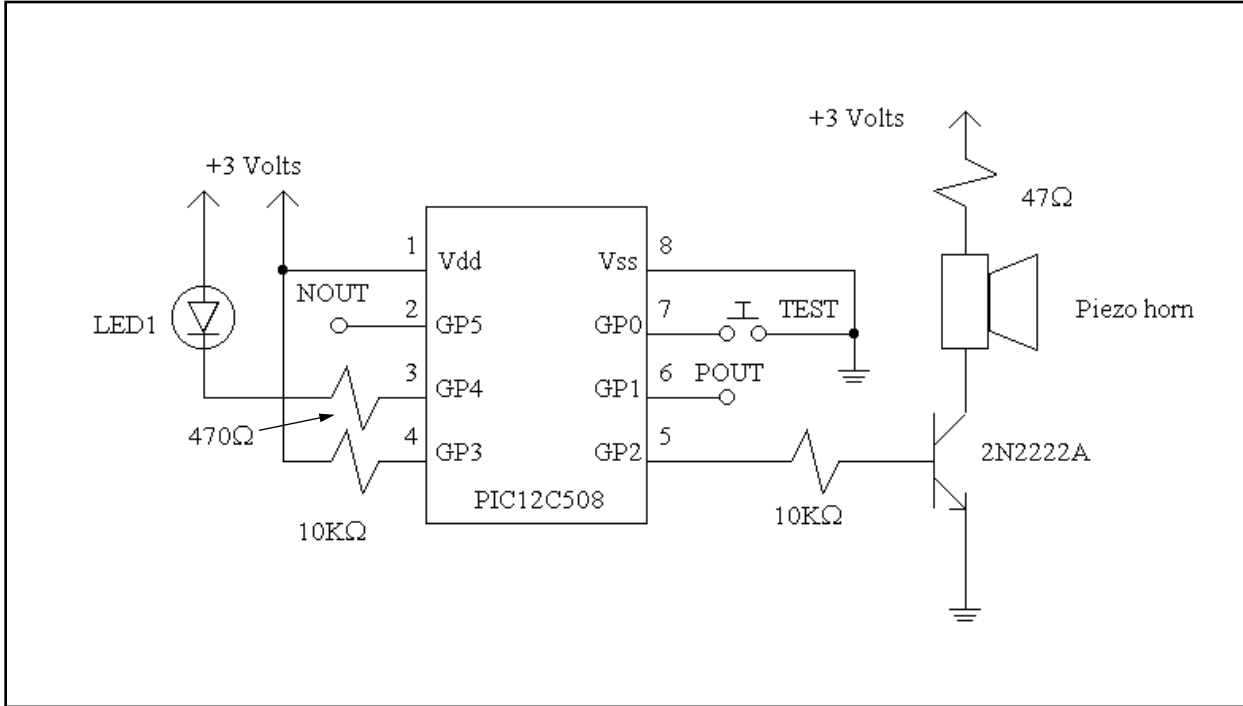
APPLICATION OPERATION

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 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.

Electromechanical Switch Replacement

GRAPHICAL HARDWARE REPRESENTATION



MICROCHIP TOOLS USED:

Development Tools:

PICSTART® Plus

Assembler/Compiler version:

MPASM for Windows®, version 1.50

Electromechanical Switch Replacement

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
;
;=====

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

;=====
;
; Register Definitions
;
;=====

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

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

INDF          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'
```

Electromechanical Switch Replacement

```
DC          EQU      H'0001'
C           EQU      H'0000'
```

```
;----- OPTION Bits -----
```

```
T0CS          EQU      H'0005'
T0SE          EQU      H'0004'
PSA           EQU      H'0003'
PS2           EQU      H'0002'
PS1           EQU      H'0001'
PS0           EQU      H'0000'
```

```
=====
```

```
;
;      RAM Definition
;
```

```
=====
```

```
    __MAXRAM H'1F'
```

```
=====
```

```
;
;      Configuration Bits
;
```

```
=====
```

```
_MCLRE_ON      EQU      H'0FFF'
_MCLRE_OFF     EQU      H'0FEF'
_CP_ON         EQU      H'0FF7'
_CP_OFF        EQU      H'0FFF'
_WDT_ON        EQU      H'0FFF'
_WDT_OFF       EQU      H'0FFB'
_LP_OSC        EQU      H'0FFC'
_XT_OSC        EQU      H'0FFD'
_IntRC_OSC     EQU      H'0FFE'
_ExtRC_OSC     EQU      H'0FFF'
```

LIST

Software listing:
(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
```

Electromechanical Switch Replacement

```
; sllep and alarm sections of the code.
;
; 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:
; Version      Date      Author
;
; 0.01         5/31/97    M R Hahn
; 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
;
;*****
;
;
; list p=PIC12C508 ;
;
; include "pl2c508.inc";
;
; _CONFIG      _INTRC_OSC & _WDT_ON & _MCLRE_ON & _CP_OFF
;
; __IDLOCS     1234h ;
;
; INDF equ 000h ;index register
; TMR0 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
;
```

Electromechanical Switch Replacement

```
;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    equ007h    ;a temporary location
Tmp1    equ 008h    ;another temp location
Delay_cnt equ009h    ;
;
SixLoequ00Bh;low byte of sixes counter
SixHiequ00Ch;hi byte of sixes counter
;
Old_statequ00Dh;
Statequ00Eh;
#definePowerupState,0;initial powerup state
#defineTestState,1;do a test beep
#defineCodeTestState,2;code test state
#defineAlarmState,3;keep track of time
;
Flagsequ00Fh;
#defineTestFlags,0;
;
fA  equ 010h;used by tdelay
fB  equ 011h;
;
Flash_cntequ012h;
Beep_cntequ013h;
;
SIXES_STARTequd'41';41 512 second periods = 6 hrs
SIXES_PWRUPequd'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    iTestgpio,0
#define    oPoutgpio,1
#define    oBeepgpio,2
#define    iResetgpio,3
#define    oLEDgpio,4
#define    oNoutgpio,5

;*****
;
; RAM Definitions
;
;*****
;
;*****
;
; VECTORS:
```


Electromechanical Switch Replacement

```
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                ;
    movlwd'200' ;
    movwfBeep_cnt;
    ;
Beep_loop          ;
    bsf  oBeep  ;
    movlwd'25'  ;
    calltdelay ;
    bcf  oBeep  ;
    movlwd'25'  ;
    calltdelay ;
    decfszBeep_cnt,f;
    gotoBeep_loop;
    ;
    retlw  0         ;return
    ;
;*****
;
; Flash:
;     Turn on LED.
;
;*****
    ;
Flash              ;
    movlwd'200' ;
    movwfFlash_cnt;
    ;
Flash_loop        ;
    bcf  oLED    ;
    callDelay    ;
    decfszFlash_cnt,f;
    gotoFlash_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
```

Electromechanical Switch Replacement

```
    tris    GPIO          ;
          ;
    movlw  b'00010000'    ;turn off all outputs
    movwf  GPIO          ;
          ;
    btfssfTest;check if in TEST mode
    gotoNoTest;no, setup long watchdog
          ;
    movlwb'01001000';very short watchdog timeout
    option ;
          ;
    gotoPassTest;
          ;
NoTest   ;
    movlw  b'01001111'    ;enable pullups, assign prescaler to WDT
    option ;              ;note: disables wake on input change
          ;
PassTest ;
    clrwdt ;clear the watch dog
          ;
    btfscOld_stat,GPWUF;check for input change caused reset
    gotoState_check;
          ;
    btfss  Old_stat,TO_FLAG;check for a watchdog timeout
    goto  State_check     ;if TO_FLAG = FALSE (WDT happened) check state
          ;
    gotoPowerup;go do powerup state
          ;
;*****
;
; State_check:
; Figure out what state we are in, and then go to the handler
; for that state.
;
;*****
;
State_check;
    btfscTest;check for test button pushed
    gotoTime_check;not pushed, check time
          ;
    btfscfTest;check if we are in test mode
    gotoTime_check;in test mode, do timekeeping
          ;
    callFlash;make some noise
    callBeep ;
    callFlash;
    callBeep ;
    callFlash;
    callBeep ;
          ;
    gotoBig_sleep;
          ;
;*****
;
; Time_check:
; Update timers, beep if it's time.
;
;*****
;
Time_check ;
    decfszSixLo,f;
    gotoNextTime;
          ;
    decfszSixHi,f;
    gotoNextTime;
          ;
```


Electromechanical Switch Replacement

```
; 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
```



Electromechanical Switch Replacement

Bright Idea Light Timer, Junior

Author: Scott A. Sumner
Eveningware, Inc.
Sterling Heights, MI
email: sasumner@bigfoot.com

APPLICATION OPERATION

Overview

The "Bright Idea" Light Timer, Jr. (BILTJR) is a digital version of the venerable lamp on/off timers that you use when you go on vacation to make it look like someone is home. I use two of these old timers (but not for much longer!) on an everyday basis, just so I don't have to turn lamps on by their switches when it gets dark and turn them off when I go to bed. The BILTJR has an advantage over the old lamp timers: it can be programmed to turn lights on and off at different times for each day of the week. It features 1 or 2 pairs of on/off times per day for 6 days with 10-minute resolution. The seventh day of the week shares its on/off times with the first day. The circuit consists of a PIC12C508 and a Dallas[®] DS1202 Serial Timekeeping chip, with very few required support components. The timer times are reprogrammable at any time using a connection to a PC's parallel port to the Dallas chip's battery-backed RAM.

Theory of Operation

The BILTJR is prepared for use by programming the on and off times for the various days of the week, as well as the current time and day of the week. This is done by connecting the circuit to a PC parallel port via the programming cable and running the programming software.

The on and off times are programmed with 10-minute resolution from midnight (0:00) to midnight of the following day (24:00). For example, if on-time #1 is set for 8 p.m. (20:00) and off-time #1 is set for 9:40 p.m. (21:40), the light output will be on between those times. The light is always extinguished as one day rolls over into the next, so programming either off-time as 24:00 will keep the light on until the day changes. The light can be kept on through midnight by programming off-time #2 for day x to be 24:00 and on-time #1 of day $x+1$ to be 0:00.

Each day of the week can have 0, 1 or 2 pairs of on/off times for the light connected to the output of the BILTJR. To have the connected light remain off for the entire day, program 24:00 for on-time #1, off-time #1, on-time #2, and off-time #2. To have the light come on and go off only one time during the day, program on-time #1 and off-time #1 with the desired times, and program 24:00 for on-time #2 and off-time #2. To have the light come on and go off two times during the day, program the desired times for on-time #1, off-time #1, on-time #2, and off-time #2.

Once the circuit is programmed, while power is applied the output will follow the programmed times. For any given day, it will be off/low before on-time #1, on/high after on-time #1 but before off-time #1, off/low after off-time #1 but before on-time #2, on/high after on-time #2 but before off-time #2, and off/low after off-time #2. The programming of the on/off times is held in non-volatile memory (battery-backed RAM) so the settings are not lost when the main power supply is removed.

The BILTJR circuit described herein has an LED and resistor for testing purposes. In a real application, the LED and resistor would be replaced by some circuitry to switch a 110 volt AC line. Also, the power supply for the circuit would also be derived from the household AC line voltage.

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.

Electromechanical Switch Replacement

HARDWARE

The BILTJR circuit consists of a PIC12C508 8-pin microcontroller, a DS1202 serial timekeeping chip, an output indicator LED, a resistor, a crystal, a battery, some diodes, some decoupling capacitors, and a cable connection header.

The light on/off output is connected to the GP5 I/O pin, and the chip select output, clock output, and data I/O lines for accessing the DS1202 are connected to GP2, GP1, and GP0, respectively. GP4 is a no-connect for now, future expansion may configure it as a light override/toggle switch input. The lines

to access the DS1202 are also brought to a connector for ease of connecting a programming cable. Ground and the PIC's GP3/MCLR input are brought to this connector as well. During programming, the PIC12C508 is held in reset by a jumper built into the programming cable so the PC parallel port (hopefully with some buffering!) can drive the DS1202 lines without interference from the PIC.

For non-volatile storage of the setup data, a 3 volt battery is used to maintain the DS1202's time and RAM storage areas. One diode prevents the battery from providing power to the PIC12C508 when the main circuit supply is down and one diode prevents the battery from presenting a load to the main supply when the supply is on. A 32.768 KHz watch crystal creates an accurate timetable for the timekeeping chip and completes the DS1202 connections.

The PIC12C508 is configured to use the internal 4MHz RC oscillator, and the GP3/MCLR pin is programmed to function as an MCLR input. For testing, it was necessary to program the on/off times using the PIC12C508 itself. Since this circuit was meant to be generic, all I/O was left as logic level. No power supply circuit was included in this circuit for the same reason; thus, an external +5V supply is necessary to power the circuit.

The test bed for the BILTJR was a PIC16C84-based circuit which will not be described in detail; however, its schematic is enclosed. The '84-based circuit is a super-set of the PIC12C508 schematic described above. It adds an RS-232 port for debugging purposes.

Software

The software was originally written for a PIC12C508 or PIC16C84 application. For ease of testing (the inevitable compile-burn-test cycle), an PIC16C84 was used for most of the testing for "Junior". That is why there are a lot of `ifdef` in the code; either the PIC12C508 or the PIC16C84 version can still be built.

The software consists of subroutines, some start-up code, and an infinite loop. The utility subroutines are for reading the clock and data areas of the DS1202 timekeeping chip and other various things such as binary-to-bcd conversions. The start-up code gets the PIC12C508 up and running and the infinite loop does the actual light timer output control. The loose flow dia-

gram below illustrates the functionality of the infinite loop. That and the well-commented source code make the program flow easy to follow for the most part.

The only obscure parts of the software are the storage of the on/off time data and the day of the week in the DS1202. This is described below:

```
Byte 0: day 1/7 on time #1
Byte 1: day 1/7 off time #1
Byte 2: day 1/7 on time #2
Byte 3: day 1/7 off time #2
Byte 4: day 2 on time #1
...
Byte 22: day 6 on time #2
Byte 23: day 6 off time #2
```

The time bytes stored in the DS1202 RAM are formatted as follows:

- HHHHHTTT (MS bit to LS bit) where HHHHH is the hour and TTT is the number of ten-minute blocks.

For example, 10:40pm is stored as b'10110100' where b'10110' is the hour (22) and b'100' is the ten-minutes (4).

- The valid range for HHHHH is b'00000' - b'11000' (0, 1, ..., 24).
- The valid range for TTT is b'000' - b'101' (0, 10, ..., 50).
- The day of the week is stored in the clock area of the DS1202 as follows:

```
Day 1: Sunday
Day 2: Monday
Day 3: Tuesday
Day 4: Wednesday
Day 5: Thursday
Day 6: Friday
Day 7: Saturday
```

Note 1: Day 7 (Saturday) duplicates the time schedule set for Day 1 (Sunday).

2: The day numbering scheme shown above is just one possible scenario; you could have Day 1 be Wednesday (and then Day 2 would be Thursday, etc., in which case Tuesday (Day 7) would be have the same on/off schedule as Wednesday (Day 1).

MICROCHIP HARDWARE DEVELOPMENT TOOLS USED

All debugging was done using the PIC16C84 test bed circuit..

Assembler/Compiler version

MPLAB 3.22.00 development software with MPASM version 1.50.

Electromechanical Switch Replacement

SOFTWARE OVERVIEW

The following is a loose description of what the software does once the PIC12C508 has come out of reset and has had its hardware registers and RAM variables initialized. Only logic concerned with the application is described and only in the most general terms for ease of understanding. Things like resetting the watchdog timer, etc., are left out for clarity.

```
(A) once per minute, do the following:
read the present time (hours, minutes, and day of week) from DS1202
subtract 1 from day of week to put it in the range 0 - 6
if day of week is 6, set the local copy to be day 0 (now day ranges 0 - 5)
if day of week has changed since last time through
    turn output off
    set state variable to be before on #1 time
    (B) calculate an index into DS1202 RAM (day of week * 4 + state)
    read DS1202 RAM location index to retrieve next output change time
    convert next change hours and ten-minutes to binary coded decimal
if state is after off #2 time, go to (A)
if the present time is equal to the next change time
    toggle the state of the output shadow bit
    advance to the next state
    if the state is not after off #2 time, go to (B)
update the real output from the output shadow bit
go to (A)
```


Electromechanical Switch Replacement

APPENDIX A: SOURCE CODE

```
;/-----\  
;| Assembler directives |  
;\-----/  
  
;comment out one or the other of the two following lines  
;    list    p=16C84      ;build code for 16C84 microcontroller  
;    list    p=12C508    ;build code for 12C508 microcontroller  
  
;    list    r=DEC       ;default radix is decimal  
;    list    x=ON       ;expand inline macros  
;    errorlevel 1,-302  ;turn off msgs caused by .inc file  
;    errorlevel 1,-205  ;turn off directive found in column 1 msgs  
  
;/-----\  
;| Assembly control #define |  
;\-----/  
  
;    ifdef __16C84  
#define DEBUG                ;include debugging code with 16C84 version  
;    endif  
  
;/-----\  
;| Processor specific include file |  
;\-----/  
  
;    ifdef __16C84  
#include "p16c84.inc"  
;    else                ;__12C508  
#include "p12c508.inc"  
;    endif  
  
;/-----\  
;| General system information |  
;\-----/  
  
;assembled using MPASM 1.50  
  
;time byte data is stored in DS1202 RAM area as follows:  
;byte 0:  day 1/7  on time #1  
;byte 1:  day 1/7  off time #1  
;byte 2:  day 1/7  on time #2  
;byte 3:  day 1/7  off time #2  
;byte 4:  day 2    on time #1  
;...  
;byte 23: day 6    off time #2  
  
;format for time byte:  
;HHHHHTTT (MS bit to LS bit) where HHHHH is the hour and TTT is the number  
; of ten-minute blocks  
;for example, 10:40pm is stored as b'10110100' where b'10110' is the  
; hour (22) and b'100' is the ten-minutes (4)  
;valid range for HHHHH is b'00000' - b'11000' (0, 1, ..., 24)  
;use b'11000' as off time #2 to keep output on until midnight  
;valid range for TTT is b'000' - b'101' (0, 10, ..., 50)  
  
;day 1:  sunday  
;day 2:  monday  
;day 3:  tuesday  
;day 4:  wednesday  
;day 5:  thursday  
;day 6:  friday  
;day 7:  saturday
```

Electromechanical Switch Replacement

```
;note: day 7 (saturday) duplicates the time schedule set for day 1 (sunday)
;note: the day numbering scheme shown above is just one possible scenario;
; you could have day 1 be wednesday (and then day 2 would be thursday, etc.,
; in which case tuesday (day 7) would be the same as wednesday (day 1)
```

```
;/-----\
;| System timing information |
;\-----/
```

```
;clock speed: 4MHz
;instruction clock speed: 4MHz / 4 = 1MHz
;time per non-branching instruction: 1us
;time per branching instruction: 2us
```

```
ITIMENS equ 1000 ;non-branching instruction time in ns
```

```
;/-----\
;| Fuses |
;\-----/
```

```
ifdef __16C84
    __config _HS_OSC & _WDT_ON & _PWRTE_ON & _CP_OFF
else
    ;__12C508
    __config _MCLRE_ON & _CP_OFF & _WDT_ON & _IntrC_OSC
endif
```

```
;/-----\
;| Miscellaneous equates, #defines, macro definitions |
;\-----/
```

```
ifdef __16C84
MINRAM equ h'0c' ;first RAM location
MAXRAM equ h'2f' ;last RAM location
MAXROM equ h'03ff' ;last program word
else
    ;__12C508
MINRAM equ h'07' ;first RAM location
MAXRAM equ h'1f' ;last RAM location
MAXROM equ h'01fe' ;last program word
endif

RESET equ h'0000' ;location where jump to on reset

#define subwl sublw ;fix for microchip's bad mnemonic
#define SUBWL sublw ; both upper and lower case

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

SKPGEZ macro ;used after a subtract instruction, this macro
    btfss STATUS,C ; will skip the next instruction if the result
    endm ; of the subtraction is >= 0

#define INSTRS (((usec*1000)/ITIMENS)-4)
DELAYUS macro usec ;this macro forms a wrapper around the delay
    ; subroutine, autocalculating the parameters
    ; needed by that subroutine using the
    ; argument to this macro (the approximate
    ; number of microseconds to delay);
```

Electromechanical Switch Replacement

```
    if INSTRS < 20
    error    "delay time is too small!"
    endif
    if INSTRS > 65535
    error    "delay time is too large!"
    endif
    if low ((INSTRS / 4) * 4) != low INSTRS
    messg    "delay will not be quite exact!"
    endif
    movlw    low INSTRS
    movwf    dlyL
    movlw    high INSTRS
    movwf    dlyH
    call     delay
    endm

;-----\
;| Microchip's one-line special instruction mnemonics |
;-----/

;CLRC, SETC, CLRDC, SETDC, CLRZ, SETZ, SKPC, SKPNC,
;SKPDC, SKPNDC, SKPZ, SKPNZ, TSTF, MOVFW
;(can use in upper or lower case)

;-----\
;| Equates for RAM variables in page 0 (h'0c' to h'2f') |
;-----/

        cblock    MINRAM
;note:  sec1202 thru wp1202 must remain in order & contiguous
sec1202                ;seconds to read/write from/to DS1202
min1202                ;minutes to read/write from/to DS1202
hr1202                 ;hours to read/write from/to DS1202
day1202                ;days to read/write from/to DS1202
mon1202                ;months to read/write from/to DS1202
dow1202                ;day of the week to read/write from/to DS1202
yr1202                 ;years to read/write from/to DS1202
wp1202                 ;write enable/disable the DS1202 clock
;note:  sec1202 thru wp1202 must remain in order & contiguous
addrEe                 ;PIC12C508 eeprom address to read/write
dataEe                 ;PIC12C508 eeprom data to write
eye                    ;loop counter variable
jay                    ;loop counter variable
kay                    ;loop variable used by clkByte subroutine
adr1202                ;address in DS1202 to read or write
dat1202                ;data value read from or to write into DS1202
temp                   ;temporary storage
bitVars                ;unrelated one-bit variables
prevMin                ;last minute value from DS1202 variable
state                  ;on#/off# state variable
chgHrs                 ;hours of next state change variable
chgMins                ;ten minutes of next state change variable
prevDay                ;variable used to detect when day changes
bcdL                   ;LSB of result of binary to BCD conversion
bcdH                   ;MSB of result of binary to BCD conversion
ENDRAM1                ;dummy value used to see if over RAM limit
        endc
        ifdef    __16C84
        cblock    ENDRAM1
txData                 ;RS-232 transmit data value
dlyH                   ;variable used by delay subroutine
dlyL                   ;variable used by delay subroutine
dlyTemp                ;variable used by delay subroutine
ENDRAM2                ;dummy value used to see if over RAM limit
```

Electromechanical Switch Replacement

```
endc
endif

ifdef __16C84

if (ENDRAM2 - 1 > MAXRAM)
error "too many RAM variables defined!"
endif

else
;__12C508

if (ENDRAM1 - 1 > MAXRAM)
error "too many RAM variables defined!"
endif

endif

;|-----\
;| I/O port bit #defines and data direction equates for port a |
;|-----/

ifdef __16C84

#define RXD232 PORTA,4 ;RS-232 receive data (i) (o.c. out./s.t. in.)
#define TXD232 PORTA,3 ;RS-232 transmit data (o)
#define DTR232_ PORTA,2 ;RS-232 data terminal ready (i)
#define UNUSED1 PORTA,1 ;unused (o)
#define LIGHT PORTA,0 ;solid state relay control to power light (o)
PORTAIO equ b'00010100' ;direction bits for port A (3 MSBs don't care)

endif

;|-----\
;| I/O port bit #defines and data direction equates for port b |
;|-----/

ifdef __16C84

#define UNUSED2 PORTB,7 ;unused (i) (weak pull-up)
#define UNUSED3 PORTB,6 ;unused (i) (w.p.u.)
#define UNUSED4 PORTB,5 ;unused (i) (w.p.u.)
#define OVERRIDE_ PORTB,4 ;override toggle pushbutton (i) (w.p.u.)
#define CS1202 PORTB,3 ;chip select for DS1202 clock chip (o)
#define IOPIN 2 ;line that is both an input and an output
#define DAT1202 PORTB,IOPIN ;serial data line to DS1202 clock chip (i/o)
#define CLK1202 PORTB,1 ;serial clock line to DS1202 clock chip (o)
#define UNUSED5 PORTB,0 ;unused (o)

PORTBIO equ b'11110100' ;direction bits for port B

if (PORTBIO < b'11000000')
error "to do in-circuit programming, rb7 and rb6 must be inputs!"
endif

endif

;|-----\
;| I/O port bit #defines and data direction equates for gpio port |
;|-----/

ifdef __12C508

#define LIGHT GPIO,5 ;solid state relay control to power light (o)
```

Electromechanical Switch Replacement

```
#define UNUSED GPIO,4 ;override toggle pushbutton (i)
#define RESET_ GPIO,3 ;reset line for PIC12C508 (i) (w.p.u)
#define CS1202 GPIO,2 ;chip select for DS1202 clock chip (o)
#define DAT1202 GPIO,1 ;serial data line to DS1202 clock chip (i/o)
#define CLK1202 GPIO,0 ;serial clock line to DS1202 clock chip (o)

DATAINP equ b'00011010' ;direction bits for gpio port (DAT1202 input)
DATAOUT equ b'00011000' ;direction bits for gpio port (DAT1202 output)

endif

;|-----\
;| Equate for option register |
;|-----/

ifdef __12C508
OPTREG equ b'11001000' ;disable wake-up, diable pull-ups, 1:1 w-dog
endif

;|-----\
;| DS1202 equates and bit definitions |
;|-----/

BURSTRD equ b'10111111' ;burst read clock portion of DS1202
BURSTWR equ b'10111110' ;burst write clock portion of DS1202

RD1202 equ 0 ;read/not write bit in DS1202 command
RAM1202 equ 6 ;RAM/not clock bit in DS1202 command

SEC1202 equ b'00000' ;DS1202 seconds register address

CTL1202 equ b'00111' ;DS1202 control register address
WEN1202 equ b'00000000' ;data to allow clock writes to DS1202
WPR1202 equ b'10000000' ;data to disallow clock writes to DS1202

;|-----\
;| Miscellaneous equates |
;|-----/

ifdef __16C84
#define CR 13 ;carriage return ASCII code
#define LF 10 ;line feed ASCII code
endif

#define PREON1 0 ;state before on time #1
#define PREOFF1 1 ;state after on time #1 but before off time #1
#define PREON2 2 ;state after off time #1 but before on time #2
#define PREOFF2 3 ;state after on time #2 but before off time #2
#define PSTOFF2 4 ;state after off time #2

;|-----\
;| bitVars bit definitions |
;|-----/

#define RAMCLK bitVars,0 ;access DS1202 RAM/not clock indicator
#define LITESH D bitVars,1 ;output on/off shadow bit

;-----\
;|-----/
```

Electromechanical Switch Replacement

```
;/ Setup reset and interrupt vectors |
;\-----/

        org      RESET            ;reset sends execution here

        ifndef __12C508
        movwf   OSCCAL            ;trim internal RC oscillator
        endif

        goto    initHW           ;assure jump over hardcoded isr

;-----/

;/-----\
;/ Routine for sending a data byte serially at 9600 baud.
;/
;/ Inputs:  w, data to send
;/
;/ Outputs: none
;/
;/ Calls:  none
;\-----/

        ifndef __16C84

send232 movwf   txData            ;save data to send

        movlw   8 + 1 + 1        ;8 bits of data, 1 start, 1 stop bit
        movwf   jay

loop232 movlw   high jmpStrt      ;get high order bits of program counter
        movwf   PCLATH           ; and save so adding to pc low works ok
        decf   jay,w

jmpStrt addwf   PCL,f            ;determine what to do and take the same
        goto   stop              ; amount of time no matter what
        FILL   (goto rotate),8
jmpEnd  goto   start

stop    goto   $ + 1             ;waste 3 cycles (includes nop at send1L1)
        goto   send1L1          ;sending a stop bit (stop bit is logic 1)

rotate  rrf     txData,f         ;figure out value of data bit to send
        SKPNC
        goto   send1L1
        goto   send0

start   goto   $ + 1             ;waste 3 cycles
        nop
        goto   send0            ;sending a start bit (start bit is logic 0)

send1L1 nop                     ;equalize inter-bit delays
send1   bsf     TXD232           ;output a 1
        goto   endLoop

send0   bcf     TXD232           ;output a 0
        goto   endLoop          ;equalize inter-bit delays

endLoop DELAYUS 86              ;104 us (1 bit time) - 18 us (loop time)

        decfsz  jay,f           ;skip next if done with data and framing bits
        goto   loop232         ;not done, go get another bit

        return
```

Electromechanical Switch Replacement

```
    if (high jmpStrt != high jmpEnd)
        error "jump table crosses page boundary in subroutine send232!"
    endif

    endif                                ;__16C84

;-----\
;| Routine for burst reading clock data from the Dallas 1202 Serial
;| Timekeeping chip.
;|
;| Inputs:  none
;|
;| Outputs: sec1202 thru wp1202
;|
;| Calls:  none
;|-----/

rdClock bsf     CS1202        ;activate the chip by selecting it
        bcf     CLK1202      ;start out with the clock low

        movlw  sec1202       ;point indirect addressing to the first byte
        movwf  FSR           ; in PIC12C508 RAM to fill

        movlw  8             ;command to DS1202 is 8 bits long
        movwf  jay

        movlw  BURSTRD       ;burst read clock data command
        movwf  sec1202

        ifdef  __16C84
        bsf   STATUS,RP0
        bcf   TRISB,IOPIN    ;make the data i/o pin an output temporarily
        bcf   STATUS,RP0
        else
        movlw DATAOUT       ;__12C508
        tris  GPIO           ;make the data i/o pin an output temporarily
        endif

rdLoop1 bcf     CLK1202      ;lower the clock

        bcf   DAT1202        ;assume command bit is going to be a 0
        rrf   sec1202,f      ;look at actual command bit
        SKPNC ;skip next if it really was 0
        bsf   DAT1202        ;not a 0, correct it to be a 1

        bsf   CLK1202        ;command data gets clocked in on rising edge

        decfsz jay,f         ;skip next if clocked in all 8 command bits
        goto  rdLoop1       ;continue clocking in command bits

        ifdef  __16C84
        bsf   STATUS,RP0     ;done outputting command to DS1202
        bsf   TRISB,IOPIN    ;revert data i/o pin back to an input
        bcf   STATUS,RP0
        else
        movlw DATAINP       ;__12C508
        tris  GPIO           ;revert data i/o pin back to an input
        endif

        movlw  8             ;we're getting 8 bytes of data from DS1202
        movwf  jay

rdLoop2 movlw  8             ;each byte is 8 bits
        movwf  kay
```

Electromechanical Switch Replacement

```
rdLoop3 bcf      CLK1202      ;clock out a data bit on clock falling edge

        CLRC                ;assume data bit is going to be a 0
btfsc   DAT1202          ;skip next if actual data bit was a 0
SETC                    ;not a 0, correct it to be a 1
rrf     INDF,f           ;rotate data bit into current PIC12C508 RAM location

        bsf      CLK1202      ;raise the clock in preparation of next bit

decfsz  kay,f           ;skip next if done with current data byte
goto    rdLoop3         ;keep working on getting current data byte

incf    FSR,f           ;point to destination for next data byte

decfsz  jay,f           ;skip next if done getting all data bytes
goto    rdLoop2         ;contine getting next data byte

bcf     CLK1202          ;leave the clock low
bcf     CS1202           ;deselect the clock chip
retlw   0
```

```
;/-----\
;| Routine for burst writing clock data to the Dallas 1202 Serial
;| Timekeeping chip.
;|
;| Inputs:  sec1202 thru wp1202
;|
;| Outputs: none
;|
;| Calls:  none
;|
;| Note:   Need to write-enable DS1202 before & write-protect it after
;|-----;
```

```
        ifndef __16C84

wrClock bsf      CS1202      ;activate the chip by selecting it
        bcf      CLK1202      ;start out with the clock low

        movlw   sec1202       ;point indirect addressing to the first byte
        movwf   FSR           ; in PIC12C508 RAM to get data from

        movlw   8             ;command to DS1202 is 8 bits long
        movwf   jay

        movlw   BURSTWR       ;burst write clock data command
        movwf   eye

        bsf     STATUS,RP0
        bcf     TRISB,IOPIN   ;make the data i/o pin an output temporarily
        bcf     STATUS,RP0

wrLoop1 bcf      CLK1202      ;lower the clock

        bcf     DAT1202       ;assume command bit is going to be a 0
        rrf     eye,f         ;look at actual command bit
        SKPNC                    ;skip next if it really was 0
        bsf     DAT1202       ;not a 0, correct it to be a 1

        bsf     CLK1202       ;command data gets clocked in on rising edge

decfsz  jay,f           ;skip next if clocked in all 8 command bits
goto    wrLoop1         ;continue clocking in command bits
```

Electromechanical Switch Replacement

```
        movlw 8           ;we're putting 8 bytes of data in the DS1202
        movwf jay

wrLoop2 movlw 8           ;each byte is 8 bits
        movwf kay

wrLoop3 bcf  CLK1202      ;lower the clock

        bcf  DAT1202      ;assume data bit is going to be a 0
        rrf  INDF,f       ;rotate data bit from current PIC12C508 RAM location
        SKPNC             ;skip next if it really was 0
        bsf  DAT1202      ;not a 0, correct it to be a 1

        bsf  CLK1202      ;clock in the data bit

        decfsz kay,f      ;skip next if done with current data byte
        goto wrLoop3     ;keep working on getting current data byte

        incf  FSR,f       ;point to destination for next data byte

        decfsz jay,f      ;skip next if done getting all data bytes
        goto wrLoop2     ;contine getting next data byte

        ifdef __16C84
        bsf  STATUS,RP0   ;done outputting command to DS1202
        bsf  TRISB,IOPIN  ;revert data i/o pin back to an input
        bcf  STATUS,RP0
        else
        movlw DATAINP    ;__12C508
        tris GPIO         ;revert data i/o pin back to an input
        endif

        bcf  CLK1202      ;leave the clock low
        bcf  CS1202       ;deselect the clock chip

        retlw 0

        endif

;-----
;| Routine for reading 1 byte of data from the Dallas 1202 Serial
;| Timekeeping chip.
;|
;| Inputs:  adr1202
;|          bitVars bit RAMNCLK (read from RAM/not clock area of DS1202)
;|
;| Outputs: dat1202
;|
;| Calls:  none
;|-----
; \

rd1202 bsf  CS1202        ;activate the chip by selecting it
        bcf  CLK1202      ;start out with the clock low

        movlw 8           ;command to DS1202 is 8 bits long
        movwf jay

        MOVFW adr1202     ;don't destroy DS1202 address
        movwf temp        ;turn address into a valid DS1202 command
        rlf  temp,f       ; byte
        bsf  temp,RD1202   ;set read/not write bit in DS1202 command
        bsf  temp,7        ;this bit is always set in valid command

        bsf  temp,RAM1202  ;assume writing to RAM area of DS1202
        btfss RAMNCLK     ;skip next if really writing to RAM area
```

Electromechanical Switch Replacement

```
    bcf     temp, RAM1202    ;really writing to clock area of DS1202

    #ifdef __16C84
    bsf     STATUS, RP0
    bcf     TRISB, IOPIN    ;make the data i/o pin an output temporarily
    bcf     STATUS, RP0
    else
    ;__12C508
    movlw   DATAOUT
    tris    GPIO            ;make the data i/o pin an output temporarily
    #endif

r12021p bcf     CLK1202     ;lower the clock

    bcf     DAT1202        ;assume command bit is going to be a 0
    rrf     temp, f        ;look at actual command bit
    SKPNC
    bcf     DAT1202        ;skip next if it really was 0
    ;not a 0, correct it to be a 1

    bsf     CLK1202        ;command data gets clocked in on rising edge

    decfsz  jay, f        ;skip next if clocked in all 8 command bits
    goto    r12021p      ;continue clocking in command bits

    #ifdef __16C84
    bsf     STATUS, RP0    ;done outputting command to DS1202
    bsf     TRISB, IOPIN   ;revert data i/o pin back to an input
    bcf     STATUS, RP0
    else
    ;__12C508
    movlw   DATAINP
    tris    GPIO            ;revert data i/o pin back to an input
    #endif

    movlw   8              ;retrieving 8 bits of data
    movwf   jay

r120212 bcf     CLK1202     ;clock out a data bit on clock falling edge

    CLRC
    btfscc DAT1202        ;assume data bit is going to be a 0
    SETC
    rrf     dat1202, f    ;skip next if actual data bit was a 0
    ;not a 0, correct it to be a 1
    ;rotate data bit into current PIC12C508 RAM

    bsf     CLK1202        ;raise the clock in preparation of next bit

    decfsz  jay, f        ;skip next if done retrieving data byte
    goto    r120212      ;keep working on getting data byte

    bcf     CLK1202        ;leave the clock low
    bcf     CS1202         ;deselect the clock chip
    retlw   0

;-----\
;| Routine for writing 1 byte of data to the Dallas 1202 Serial Timekeeping |
;| chip. |
;| |
;| Inputs:  adr1202 (the address in the DS1202 to write) |
;|          dat1202 (the data to write to the specified address) |
;| |
;|          bitVars bit RAMNCLK (write to RAM/not clock area of DS1202) |
;| |
;| Outputs: none |
;| |
;| Calls:  none |
;| |
;| Note:  Need to write-enable DS1202 before & write-protect it after |
;|-----/
```

Electromechanical Switch Replacement

```
;\-----/

    ifdef __16C84

wr1202 bsf     CS1202      ;activate the chip by selecting it
      bcf     CLK1202    ;start out with the clock low

      movlw  16          ;command & data to DS1202 are each 8 bits
      movwf  jay

      MOVFW  adr1202     ;don't destroy DS1202 address
      movwf  temp       ;turn address into a valid DS1202 command
      rlf   temp,f      ; byte
      bcf   temp,RD1202 ;clear read/not write bit in DS1202 command
      bsf   temp,7      ;this bit is always set in valid command

      bsf   temp,RAM1202 ;assume writing to RAM area of DS1202
      btfss RAMCLK      ;skip next if really writing to RAM area
      bcf   temp,RAM1202 ;really writing to clock area of DS1202

      bsf   STATUS,RP0
      bcf   TRISB,IOPIN ;make the data i/o pin an output temporarily
      bcf   STATUS,RP0

w1202lp bcf     CLK1202    ;lower the clock

      bcf   DAT1202     ;assume command bit is going to be a 0
      rrf   temp,f      ;look at actual command bit
      SKPNC ;skip next if it really was 0
      bsf   DAT1202     ;not a 0, correct it to be a 1

      bsf   CLK1202     ;command data gets clocked in on rising edge

      movlw  9          ;jay will be 9 when we've clocked out 8 bits
      xorwf  jay,w      ;skip next if done with 8 bit command
      SKPZ  ;skip next if done with 8 bit command
      goto  w1202ov     ;keep working on command bits
      MOVFW  dat1202    ;done with command bits, switch to data bits
      movwf  temp

w1202ov decfsz  jay,f     ;skip next if clocked in all 16 bits
      goto  w1202lp    ;continue clocking in command bits

      bsf   STATUS,RP0  ;done outputting command to DS1202
      bsf   TRISB,IOPIN ;revert data i/o pin back to an input
      bcf   STATUS,RP0

      bcf   CLK1202    ;leave the clock low
      bcf   CS1202     ;deselect the clock chip
      retlw  0

      endif

;\-----/
;| Routine for converting a BCD digit (0 - 9) to ASCII.
;|
;| Inputs:  w (the BCD digit (only lower nibble is relevant))
;|
;| Outputs: w (the converted ASCII code)
;|
;| Calls:  none
;\-----/

    ifdef __16C84
```

Electromechanical Switch Replacement

```
bcd2asc andlw  h'0f'          ;clear upper nibble
        iorlw  h'30'          ;set bits 4 & 5 to make valid ASCII code
        return

        endif                ;__16C84

;-----\
;| Routine for converting a 1-byte binary value to a 2-byte binary-coded
;| decimal value (2 digits) (taken from AN526 "PIC12C508 16C5X/16CXX
;| Math Utility Routines" from Microchip .
;| Embedded Control Handbook, page 5-119)
;|
;| Inputs:  w, the binary value to convert (h'00'-h'63')
;|
;| Outputs: bcdH,bcdL
;|
;| Calls:  none
;|-----\

bin2bcd clrf   bcdH
        movwf bcdL
gtenth  movlw  10
        subwf bcdL,w
        btfss STATUS,C
        goto  endBcd
        movwf bcdL
        incf  bcdH,f
        goto  gtenth
endBcd  return

;-----\
;| Routine for generating a programmable delay (routine written by Philip
;| Doucet - obtained from Electronics Design - August 8, 1994, page 26ES)
;| This "delay" subroutine requires three registers. The 16-bit argument
;| is in dlyH and dlyL. Minimum value of the argument is 20. Register
;| dlyTemp is needed for temporary storage. This routine will delay 20
;| or more instruction cycles. For exact accuracy, the delay parameter
;| must be a multiple of 4.
;|
;| Inputs:  # of instructions to delay in dlyL and dlyH
;|
;| Outputs: none
;|
;| Calls:  none
;|-----\

        ifdef __16C84

delay   movlw  20              ;subtract minimum # of instructions to
        subwf  dlyL,f          ;execute this routine from requested delay
        SKPC                    ;check for borrow & decrement high byte if
        decf  dlyH,f          ;there was one
        CLRC                    ;divide by 4 to determine how many times to
        rrf   dlyL,f          ;execute dlyL loop
        CLRC
        rrf   dlyL,f
        movf  dlyH,f          ;check to see if dlyH = 0 & skip dlyH loop if
        SKPNZ                    ;it is
        goto  delay3
        nop                    ;nop equalizes timing between paths
delay1  movlw  62              ;since each dlyH loop needs 256 cycle, or 64
        movwf dlyTemp         ;times thru inner loop of cycles, minus
        nop                    ;cycle setup, so 64 - 2 = 62
        goto  delay2          ;add a 2 cycle delay
```

Electromechanical Switch Replacement

```
delay2  nop                ;inner loop for dlyH
        decfsz dlyTemp,f
        goto   delay2
        nop
        decfsz dlyH,f      ;outer loop for dlyH
        goto   delay1
        nop
delay3  movf   dlyL,f      ;if dlyL = 0, skip loop
        SKPNZ
        goto   dlyEnd
        nop
delay4  nop                ;loop for dlyL
        decfsz dlyL,f
        goto   delay4
        nop
dlyEnd  return            ;return from subroutine

        endif            ;__16C84

;-----

;Do PIC12C508 initialization here, including setting up I/O and configuring control
; registers. Timer 0 is set up as a timer to drive the application clock at
; 64 ticks per second and to blink the LEDs when necessary. ;Clear system
; interrupt flags, and enable interrupts (they are disabled on reset or
; powerup). Other initialization is self-explanatory.

initHW

        ifdef __16C84

        clrf   PORTA      ;set port output latches to a known state
        clrf   PORTB

        bcf    INTCON,GIE ;disable all interrupt sources

        bcf    EEADR,7    ;avoid higher than necessary current drain
        bcf    EEADR,6

        bsf    STATUS,RP0 ;select page 1 (powerup default is page 0)

        bcf    OPTION_REG,NOT_RBPU ;enable weak pullups on port B

        bsf    OPTION_REG,T0CS ;select external source for timer 0
        bsf    OPTION_REG,T0SE ;select falling edge as timer 0 increment

        bsf    OPTION_REG,PSA ;assign prescaler to watchdog timer
        bcf    OPTION_REG,PS2
        bcf    OPTION_REG,PS1
        bcf    OPTION_REG,PS0 ;1:1 prescale watchdog timer (18 ms)

        MOVFW  TRISA
        andlw  b'11100000'
        iorlw  PORTAIO;port A input/output pin configuration
        movwf  TRISA      ; (leave 3 most-significant bits alone)

        movlw  PORTBIO    ;port B input/output pin configuration
        movwf  TRISE

        bcf    STATUS,RP0 ;set default page back to 0

        else

        ;__12C508

        clrf   GPIO      ;set port output latches to a known state
```

Electromechanical Switch Replacement

```
        movlw   OPTREG
        option           ;disable wake-up, disable pull-ups, 1:1 w-dog

        movlw   DATAINP
        tris    GPIO           ;gpio port input/output pin configuration

        endif

endHW

;-----

;Set up initial variables and define initial conditions here.

initSW  movlw   h'ff'
        movwf   prevDay       ;initialize to an invalid value
        movwf   prevMin       ;initialize to an invalid value

        bcf     CS1202         ;make sure DS1202 is deselected

        movlw   WPR1202       ;this variable never changes; it is needed
        movwf   wp1202        ; for the burst write

        bcf     LITESHD       ;shadow bit for output starts out off
        bcf     LIGHT         ;make sure light output starts out off

        ifdef   __16C84
        bsf     TXD232         ;set RS-232 transmit line to marking state
        endif

endSW

;-----

        ifdef   DEBUG

        movlw   CTL1202       ;hard program DS1202 with '84 instead of
        movwf   adr1202       ; using the PC's parallel port for easier
        movlw   WEN1202       ; debugging
        movwf   dat1202
        bcf     RAMNCLK
        call    wr1202         ;allow writes to DS1202

        movlw   h'45'
        movwf   sec1202

        movlw   h'58'
        movwf   min1202

        movlw   h'23'
        movwf   hr1202

        movlw   h'20'
        movwf   day1202

        movlw   h'11'
        movwf   mon1202

        movlw   h'1'
        movwf   dow1202
```

Electromechanical Switch Replacement

```
movlw h'96'  
movwf yr1202  
  
call wrClock ;initialize clock time in DS1202  
  
movlw 0  
movwf adr1202  
movlw h'ba'  
movwf dat1202  
bsf RAMNCLK  
call wr1202  
  
movlw 1  
movwf adr1202  
movlw h'bb'  
movwf dat1202  
bsf RAMNCLK  
call wr1202  
  
movlw 2  
movwf adr1202  
movlw h'bc'  
movwf dat1202  
bsf RAMNCLK  
call wr1202  
  
movlw 3  
movwf adr1202  
movlw h'bd'  
movwf dat1202  
bsf RAMNCLK  
call wr1202  
  
movlw 4  
movwf adr1202  
movlw h'0'  
movwf dat1202  
bsf RAMNCLK  
call wr1202  
  
movlw 5  
movwf adr1202  
movlw h'1'  
movwf dat1202  
bsf RAMNCLK  
call wr1202  
  
movlw 6  
movwf adr1202  
movlw h'2'  
movwf dat1202  
bsf RAMNCLK  
call wr1202  
  
movlw 7  
movwf adr1202  
movlw h'3'  
movwf dat1202  
bsf RAMNCLK  
call wr1202  
  
movlw CTL1202  
movwf adr1202  
movlw WPR1202  
movwf dat1202
```

Electromechanical Switch Replacement

```
    bcf     RAMNCLK
    call    wr1202          ;disallow writes to DS1202

    endif                                ;DEBUG

infLoop clrwdt              ;pet the dog to keep him happy

    call    rdClock        ;get current day and time info

    MOVFW  prevMin         ;retrieve old minute data
    xorwf  min1202,w       ;compare to current minute data
    SKPNZ  ;skip next if minute has changed
    goto   infLoop        ;we're still in the same minute

    MOVFW  min1202         ;minute has changed
    movwf  prevMin        ;update old minute so we remember next time

    ifdef  DEBUG
    MOVFW  hr1202          ;output HH:MM to RS-232 port
    movwf  bcdL
    rrf    bcdL,f
    rrf    bcdL,f
    rrf    bcdL,f
    rrf    bcdL,w
    call   bcd2asc
    call   send232
    MOVFW  hr1202
    call   bcd2asc
    call   send232
    movlw  ':'
    call   send232
    MOVFW  min1202
    movwf  bcdL
    rrf    bcdL,f
    rrf    bcdL,f
    rrf    bcdL,f
    rrf    bcdL,w
    call   bcd2asc
    call   send232
    MOVFW  min1202
    call   bcd2asc
    call   send232
    movlw  CR
    call   send232
    movlw  LF
    call   send232
    endif

    decf   dow1202,f       ;convert day with range 1 - 7 to 0 - 6
    movlw  6
    xorwf  dow1202,w       ;compare current day of week with 6
    SKPNZ  ;skip next if not 6
    clrf   dow1202        ;wrap day 6 to be the same as day 0

    MOVFW  dow1202        ;retrieve day of week in range 0 - 5
    xorwf  prevDay,w      ;has the day changed on us?
    SKPNZ  ;skip next if it has
    goto   sameDay

    MOVFW  dow1202        ;day changed
    movwf  prevDay        ;set previous day variable to same as current

    movlw  PREON1        ;since day changed we are before on time #1
    movwf  state          ;remember that

    bcf    LITESHD        ;turn output device off
```

Electromechanical Switch Replacement

```
rdNxChg MOVFW   dowl202      ;calculate next output transition time
        movwf   temp
        CLRC
        rlf    temp,f
        CLRC
        rlf    temp,f      ;calculate day of week * 4

        MOVFW   state
        addwf   temp,w      ;w = day of week * 4 + state
        movwf   adr1202     ;store index as RAM location to read in DS1202

        bsf    RAMNCLK
        call   rd1202      ;read DS1202 RAM location [day * 4 + state]

        MOVFW   dat1202     ;retrieve HHHHHTTT binary data
        movwf   temp
        rrf    temp,f      ;rotate to get CHHHHHTT
        rrf    temp,f      ;rotate to get CCHHHHHT
        rrf    temp,w      ;rotate to get CCCHHHHH
        andlw  b'00011111'  ;mask to get 000HHHHH
        call   bin2bcd     ;convert to 2 BCD digits
        rlf    bcdH,f      ;rotate MS BCD digit to get 00MMMMC
        rlf    bcdH,f      ;rotate MS BCD digit to get 00MMMMCC
        rlf    bcdH,f      ;rotate MS BCD digit to get 0MMMMCCC
        rlf    bcdH,w      ;rotate MS BCD digit to get MMMMCCCC
        andlw  b'11110000'  ;mask to get MMMM0000
        iorwf  bcdL,w      ;combine to get MMMLLLLL hours
        movwf  chgHrs      ;save for comparison to current hours later

        clrf    temp        ;clear the addition accumulator
        MOVFW   dat1202     ;retrieve HHHHHTTT binary data
        andlw  b'00000111'  ;mask out hours to get 00000TTT
        movwf  eye
        movlw  10
        addLoop TSTF   eye  ;i = TTT = number of ten minute blocks
                        ;add 10 to accumulated sum each time thru
                        ;i down to 0 yet?
        SKPNZ
        goto   addDone     ;skip next if i > 0
                        ;i down to 0, now have minutes calculated
        addwf  temp,f      ;sum = sum + 10
        decf  eye,f        ;i = i - 1
        goto  addLoop     ;continue adding
addDone MOVFW   temp        ;retrieve minutes
        call   bin2bcd     ;convert to 2 BCD digits
        rlf    bcdH,f      ;rotate MS BCD digit to get 00MMMMC
        rlf    bcdH,f      ;rotate MS BCD digit to get 00MMMMCC
        rlf    bcdH,f      ;rotate MS BCD digit to get 0MMMMCCC
        rlf    bcdH,w      ;rotate MS BCD digit to get MMMMCCCC
        andlw  b'11110000'  ;mask to get MMMM0000
        iorwf  bcdL,w      ;combine to get MMMLLLLL minutes
        movwf  chgMins     ;save for comparison to current minutes later

sameDay MOVFW   state        ;retrieve current state
        xorlw  PSTOFF2     ;is current state after off time #2?
        SKPNZ
        goto   infLoop     ;skip next if not
                        ;time is after 2nd turn off time, recycle

chkTime MOVFW   chgHrs
        subwf  hrl202,w     ;w = current hours - next change hours
        SKPGEZ
        goto  updShdw      ;skip next if current >= next change
                        ;go update output from shadow bit
        SKPZ
        goto  change       ;skip next if current equals change
                        ;go toggle shadow output bit
        MOVFW  chgMins
        subwf  min1202,w    ;w = current minutes - next change ten minutes
        SKPGEZ
        goto  updShdw      ;skip next if current >= next change
                        ;go update output from shadow bit
```

Electromechanical Switch Replacement

```

        SKPZ          ;skip next if current equals next change
        goto         updShdw      ;go update output from shadow bit

change  btfss        LITESHD      ;skip next if shadow bit is currently on
        goto         shdwOn       ;shadow bit off, go turn it on
        bcf          LITESHD      ;turn shadow bit off
        goto         endChg       ;done with shadow bit, get out of this section
shdwOn  bsf          LITESHD      ;turn shadow bit on
endChg

        incf         state,f      ;advance to next state in the on/off machine

        MOVFW        state
        xorlw        PSTOFF2      ;are we now after off time #2?
        SKPZ          ;skip next if we are
        goto         rdNxChg      ;haven't turned off for the last time today;
        ; need to read the next time for state change

updShdw btfss        LITESHD      ;skip next if shadow says output should be on
        goto         outOff       ;output should be off so go make it so
        bsf          LIGHT        ;turn output on
        goto         endUpdt      ;done with output, get out of this section
outOff  bcf          LIGHT        ;turn output off
endUpdt

        goto         infLoop      ;repeat ad nauseum

;-----

;|-----\
;| End of program watchdog fill |
;|-----/

endProg fill      (goto wdReset),(MAXROM - $)
        org        MAXROM        ;set breakpoints on endProg thru wdtRst
wdReset goto      wdReset        ;force watchdog to fire

;-----

;|-----\
;| End assembly |
;|-----/

        end
```

Electromechanical Switch Replacement

NOTES:

Connecting Sensor Buttons to PIC12CXXX MCUs

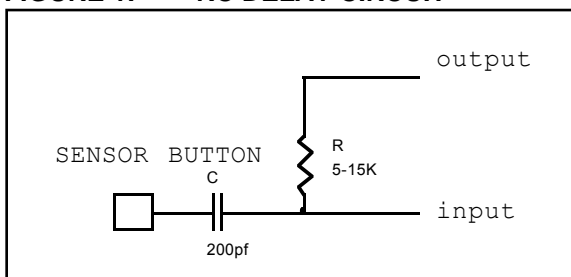
Author: Vladimir Velchev
 AVEX
 Sofia, Bulgaria



APPLICATION OPERATION

The idea is to replace the electromechanical switches with capacitive sensor buttons (PCB round or square pads). PIC12CXXX are very suitable for this purpose and with few components we may design a low cost fully electronic switch or regulator. The method uses a simple RC delay circuit (Figure 1), when the time constant changes when the sensor is touched.

FIGURE 1: RC DELAY CIRCUIT



To read the state of the sensor, the microcontroller must perform only two steps (Figures 2 and 3):

- Step 1: Changing the state of output from "0" to "1" (write operation - T_{wr}).
- Step 2: Reading the state of input (read operation - T_{rd}).

FIGURE 2: UNTOUCHED SENSOR

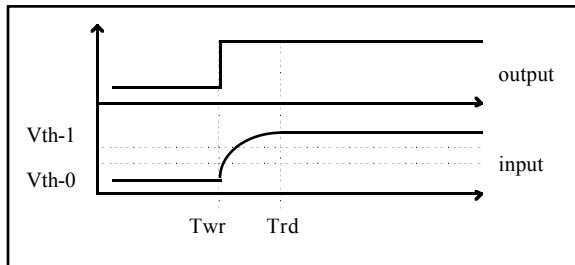
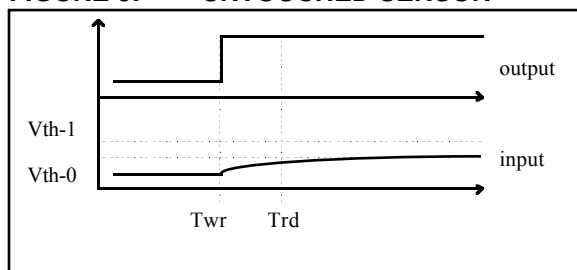


FIGURE 3: UNTOUCHED SENSOR



If the result of read operation is "0", it means that the sensor is touched. Human capacitance has been connected serially to capacitor C and the time constant of the circuit has become bigger.

Because of the small capacitance of human's fingers, the interval time between T_{wr} and T_{rd} should be less than 1-2 μs , so we recommend the use of 4 MHz internal clock of PIC12CXXX.

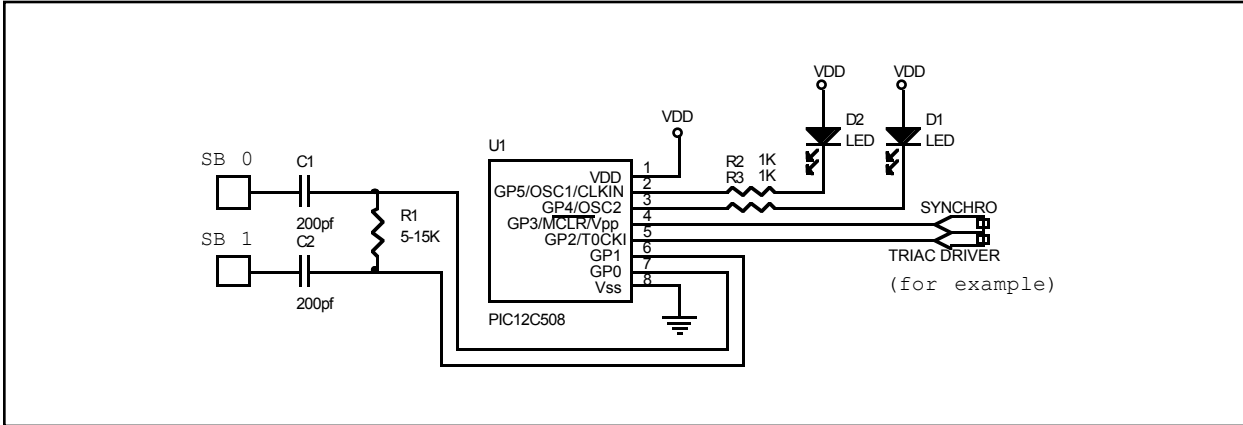
The figure under Graphical Hardware Representation contains an example with two sensor buttons. The trick here is that for reading the sensor SB0: GP0 is set as input, GP1 is set as output and for reading the SB1: GP0 is set as output, GP1 is set as input. The outputs GP4,5 are connected to LEDs and show the state of sensor buttons. The other pins are not discussed here and they can be used for example as SYNCRO input and TRIAC DRIVER output.

The value of resistor R1 must be adjusted and it determines the sensibility of the buttons. The values of capacitors C1 and C2 are not so important. We recommend the use of the same type inputs of microcontroller, with equal input impedances.

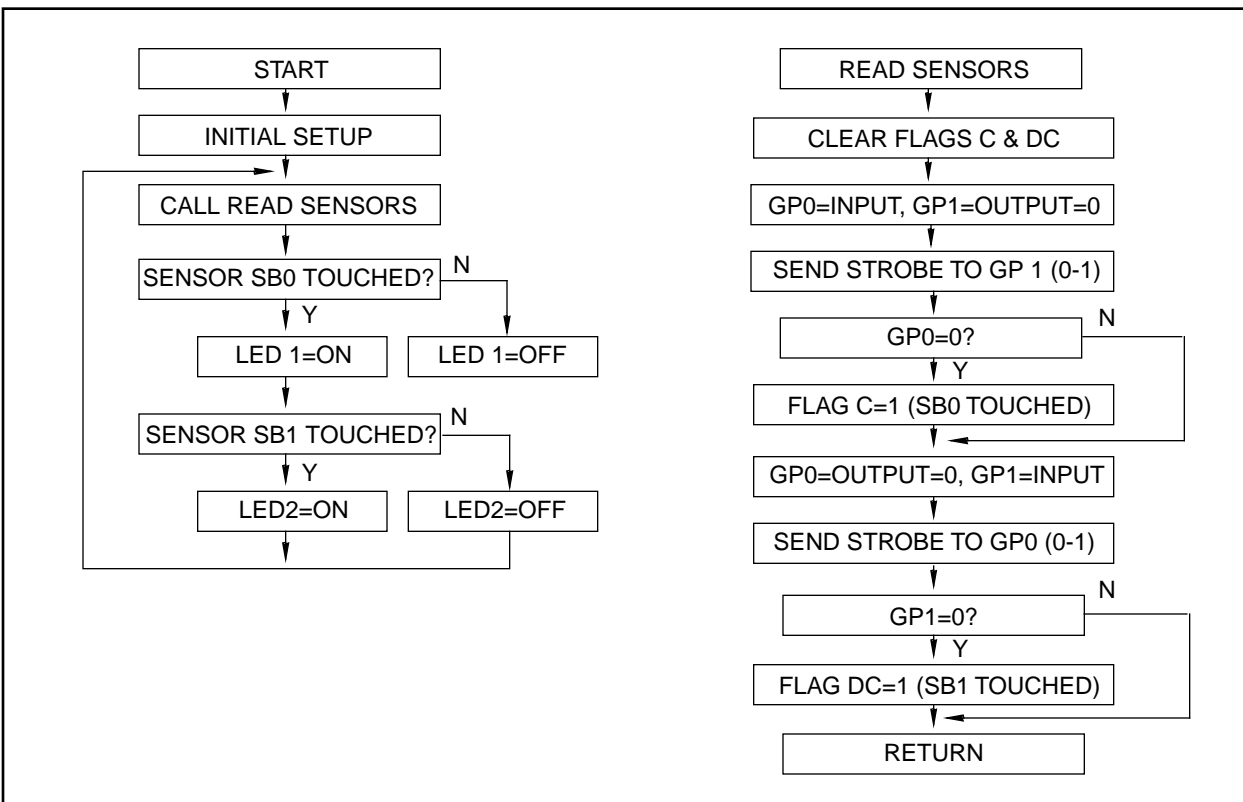
Microchip Technology Incorporated, has been granted a nonexclusive, worldwide license to reproduce, publish and distribute all submitted materials, in either original or edited form. The author has affirmed that this work is an original, unpublished work and that he/she owns all rights to such work.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

Electromechanical Switch Replacement

GRAPHICAL HARDWARE REPRESENTATION



FLOWCHART



MICROCHIP TOOLS USED

Assembler/Compiler Version:

MPLAB 3.22, MPASM 1.5

Electromechanical Switch Replacement

APPENDIX A: SOURCE CODE

```
;Software listing: sensor.asm
;*****
; Connecting sensor buttons to PIC12C508
; Osc.: F=4MHz (internal)
; Written by Vladimir Velchev 06.1997. (C) AVEX
; Version 1.00
;*****

; GP0 - sensor button SB0 (input/output)
; GP1 - sensor button SB1 (input/output)
; GP2 - not used (output)
; GP3 - not used (input)
; GP4 - LED1 for sensor button SB0 (output: 0=LED ON, 1=LED OFF)
; GP5 - LED2 for sensor button SB1 (output: 0=LED ON, 1=LED OFF)

                LIST      P=12C508

#include <p12C508.inc>

;*** Equates
SB0              equ      0              ;sensor button SB0
SB1              equ      1              ;sensor button SB1
SB0_MASK        equ      B'00000001'    ;bit mask for SB0
SB1_MASK        equ      B'00000010'    ;bit mask for SB1
OUT0            equ      0              ;driving output for SB1
OUT1            equ      1              ;driving output for SB0
LED1            equ      4              ;LED for button SB0
LED2            equ      5              ;LED for button SB1
IOSET           equ      B'00001000'    ;initial I/O port settings
                                           ;GP3-input, others- outputs

                org      0              ;RESET vector

;*** Code Starting Point
MAIN:
; Initial setup
                movlw   IOSET           ;init GPIO
                tris   GPIO
                clrf   GPIO           ;reset all outputs (=0)

Main_Loop:
                clrwdt                ;clear watchdog timer

; Space for user code
;
;
;
                call   READ_SENSORS    ;call subroutine for sensors
                btfss  STATUS,C        ;skip if sensor SB0 touched (C=1)
                goto   SB0_untouched   ;else - go to turn off LED1
                bcf    GPIO,LED1       ;LED1=ON
                goto   Check_SB1       ;go to checking next button
SB0_untouched:bsf  GPIO,LED1           ;LED1=OFF
Check_SB1:  btfss  STATUS,DC          ;skip if sensor SB1 touched (DC=1)
                goto   SB1_untouched   ;else - go to turn off LED2
                bcf    GPIO,LED2       ;LED2=ON
                goto   Main_Loop        ;go to beginning
SB1_untouched:bsf  GPIO,LED2           ;LED2=OFF
                goto   Main_Loop        ;go to beginning

;*** Subroutine - READ_SENSORS
; Input :
; Output: Flags: C=1 if sensor SB0 touched, DC=1 if sensor SB1 touched
READ_SENSORS:
                bcf    STATUS,C         ;clear carry flag
                bcf    STATUS,DC        ;clear digit carry flag
```

Electromechanical Switch Replacement

```
;Read sensor button SB0
    movlw        IOSET|SB0_MASK ;set SB0 as input (GP0)
    tris        GPIO
    bsf         GPIO,OUT1      ;send strobe to output ___---
    btfss       GPIO,SB0      ;skip if sensor SB0 untouched (GP0=1)
    bsf         STATUS,C      ;else - set C flag
;Discharging the capacitance of the input
    movlw       IOSET&(~SB0_MASK) ;set SB0 as output (GP0)
    tris       GPIO
    bcf        GPIO,OUT0      ;reset output 0 (discharging)
    bcf        GPIO,OUT1      ;reset output 1
;Read sensor button SB1
    movlw       IOSET|SB1_MASK ;set SB1 as input (GP1)
    tris       GPIO
    bsf        GPIO,OUT0      ;send strobe to output ___---
    btfss      GPIO,SB1      ;skip if sensor SB1 untouched (GP1=1)
    bsf        STATUS,DC     ;else - set DC flag
;Discharging the capacitance of the input
    movlw      IOSET&(~SB1_MASK) ;set SB1 as output (GP1)
    tris      GPIO
    bcf       GPIO,OUT1      ;reset output 1 (discharging)
    bcf       GPIO,OUT0      ;reset output 0
    return
    end                    ;end of program
```

Electromechanical Switch Replacement

NOTES:

Electromechanical Switch Replacement

NOTES:

Electromechanical Switch Replacement

NOTES:



Electromechanical Switch Replacement

Electronic Key, Button Dimmer and Potentiometer Dimmer Controller

Author: Slav Slavov
EII
Sliven, Bulgaria
email: ell@sliven.osf.acad.bg

APPLICATION OPERATION

These three applications are designed to replace AC electromechanical keys. **They must not be used for DC because of the use of triacs.** All of the applications may be powered directly from AC.

Application 1: Electronic Key

The application shown in Figure 1 can replace almost all electromechanical keys. It is synchronized with the line voltage so the charge is switched on only in the beginning of the half period. It may be used in drills, fans and many electrical machines used at home.

As you can see from Figure 1, the program makes a loop, where it is first waiting for the beginning of the first half period. When this is reached, the button is tested. If it is pushed, it switches on the triac else does not switch on the triac. After that, it waits for the beginning of the second half period and so on.

Application 2: Button Dimmer Controller

The application shown in Figure 2 may be used as a dimmer or revolution controller. Button 1 increases power and button 2 decreases power. To make the input/output linear, I am using a table to convert the input value. The half period is divided to 64 areas the surface of which are equal:

$$\int_{X_n}^{X(n+1)} \sin(x) = \int_{X(n+1)}^{X(n+2)} \sin(x)$$

The TMR0 is cleared every time at the beginning of the half period. The Prescaler of the timer is set to divide by 64 so when $F_{osc} = 4\text{MHz}$, the value that will be in TMR0 at the end of the half period will be:

$$\frac{10\text{ms}(\text{half period for } 50\text{Hz net})}{64 \times 1\mu\text{s}} = 156$$

The maximum value that is returned from the table is 154. So there is a cycle that compares the value of the timer and the value returned from the table. When the value of the timer goes greater than the output, it is activated, and, in the rest of the time to the end of the half period the buttons are read. If the controller is set to minimum, there will not be time for button read. That's why the table doesn't return 156 as maximum but 154. This is not very important in the beginning and in the end of the half period.

When the button is pushed, an action is performed (increase or decrease). If the button is still held down after about 1 second, the second action will be performed. After that, the time to the next action is about 0.1 second. If the max/min value is reached, no action is performed.

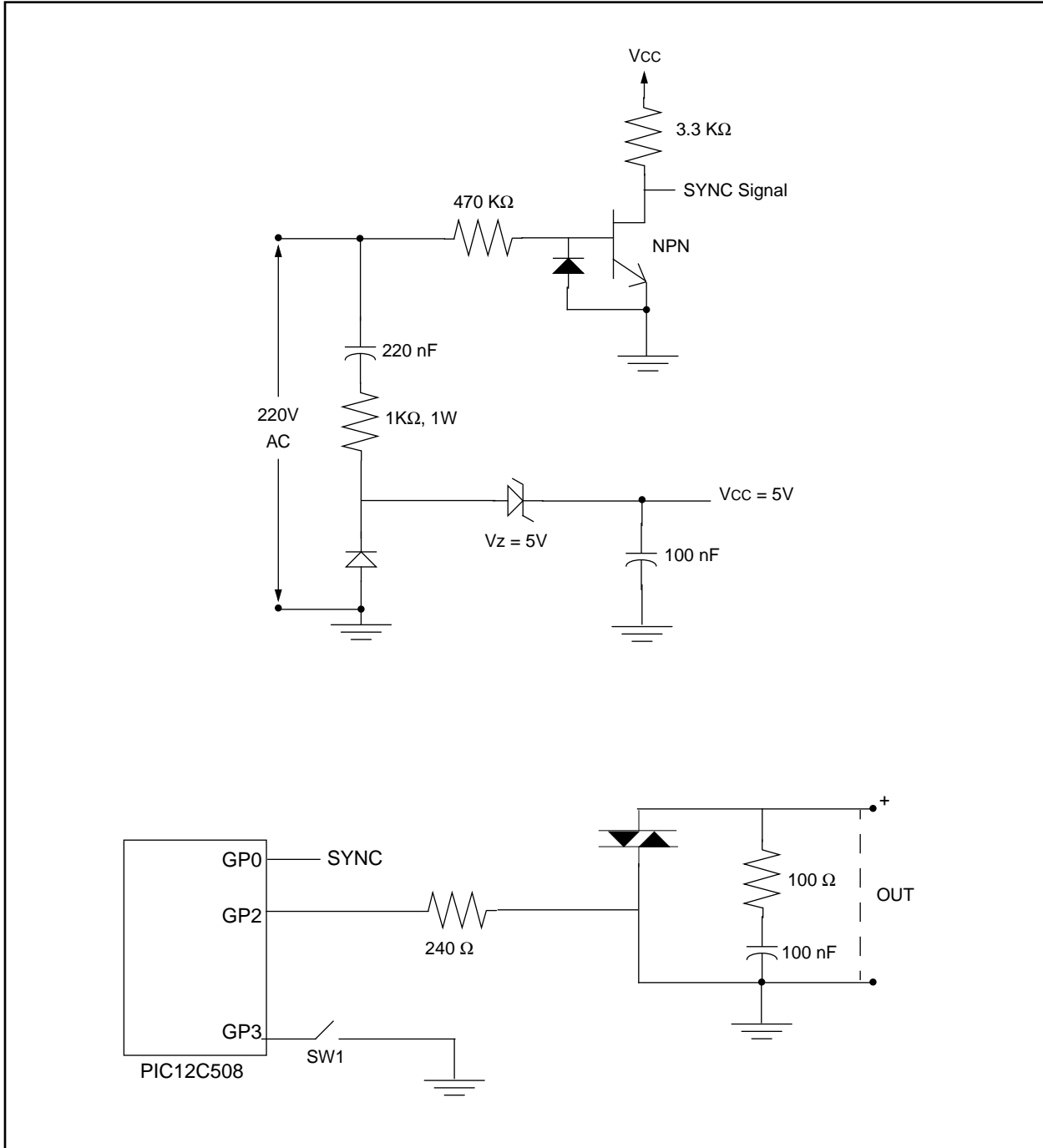
Application 3: Potentiometer Dimmer Controller

The application shown in Figure 3 is similar to application 2, but there is a potentiometer instead of buttons. This uses the A/D converter from the PIC12C671. The A/D conversion is started in the end of the half period and the ADRES register is read when the GO/DOWN bit is down.

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.

Electromechanical Switch Replacement

FIGURE 1: POWERING WITHOUT TRANSFORMER



Electromechanical Switch Replacement

FIGURE 2: BUTTON DIMMER CONTROLLER

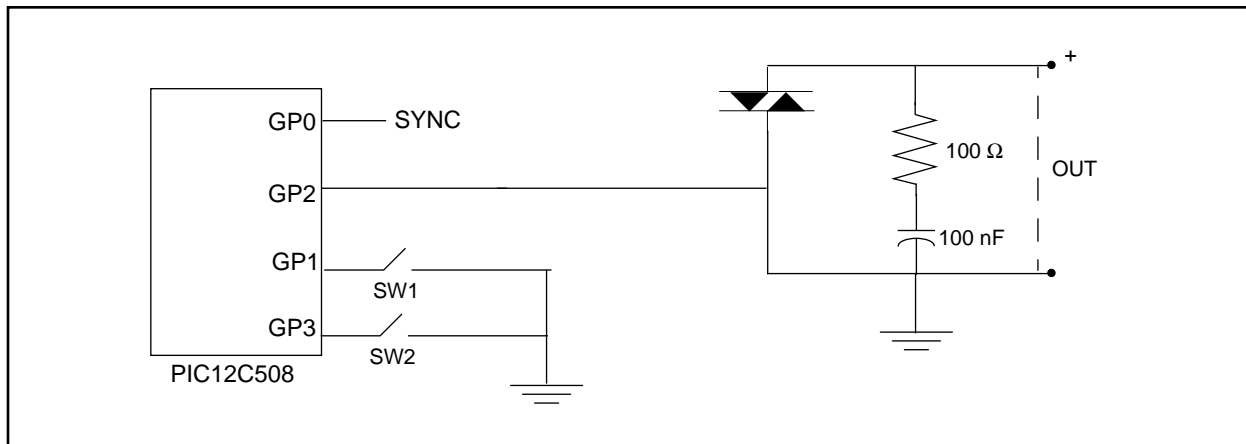
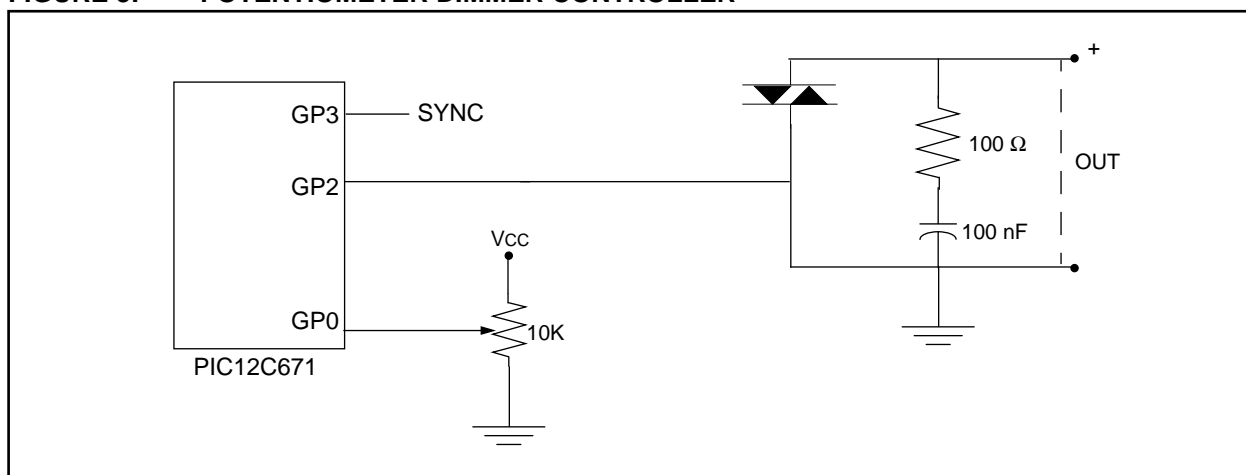
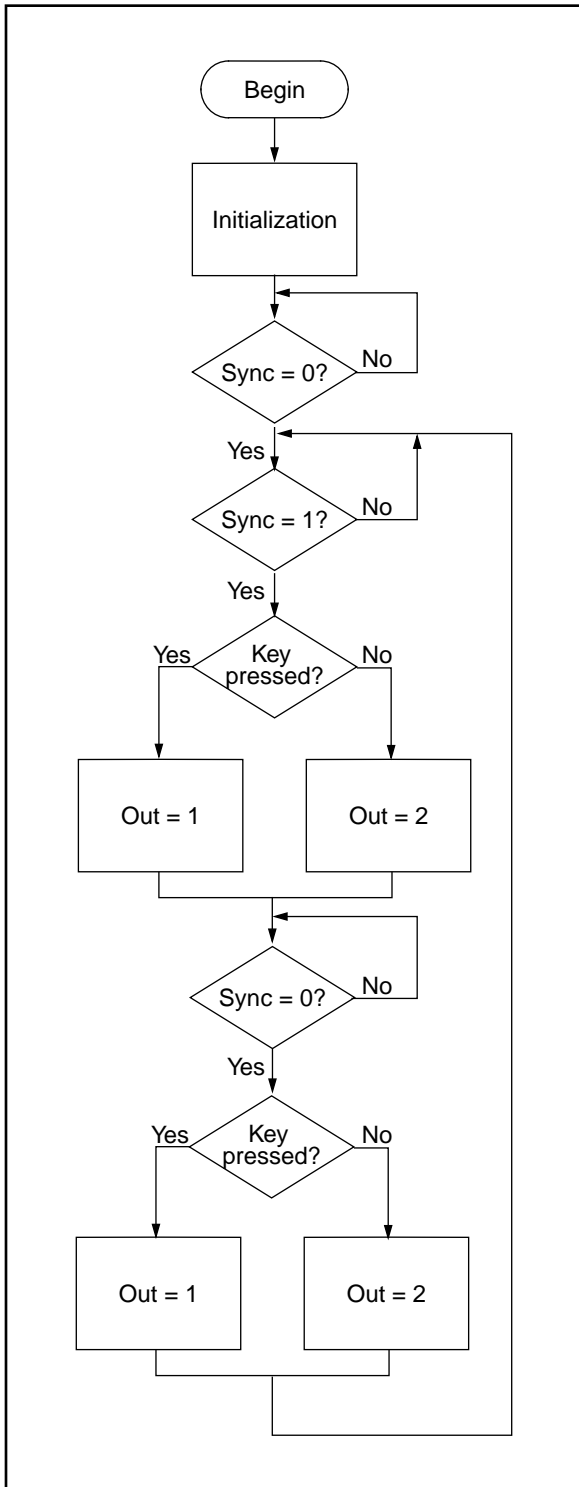


FIGURE 3: POTENTIOMETER DIMMER CONTROLLER

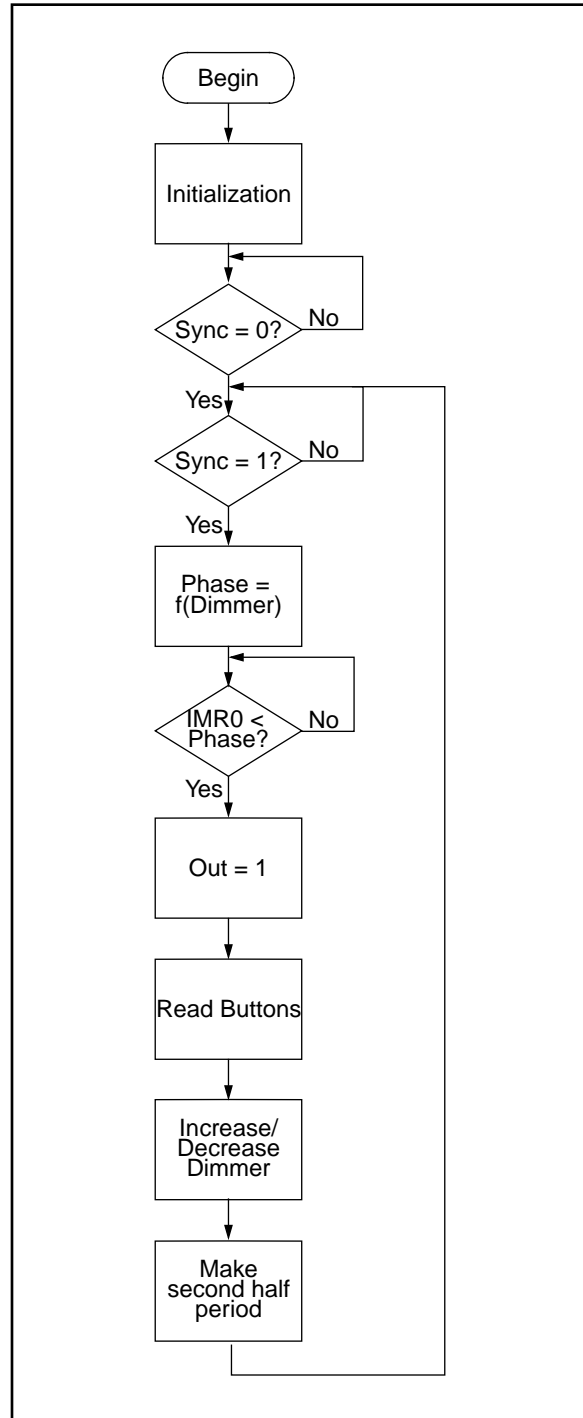


Electromechanical Switch Replacement

**FIGURE 4: APPLICATION 1
ELECTRONIC KEY**



**FIGURE 5: APPLICATION 2 AND 3
BUTTON DIMMER AND
POTENTIOMETER
CONTROLLERS**



MICROCHIP TOOLS USED:

Development Tools:

PICSTART® Plus V1.20

Assembler/Compiler Version:

MPLAB V3.22, MPASM V1.5

Electromechanical Switch Replacement

APPENDIX A: SOURCE CODE

```
*****
;
;                               Figure1.ASM
;
*****

LIST    p=12C508

#include    "inc\p12c508.inc"

        __config    _WDT_ON & _IntrC_OSC & _MCLRE_OFF & _CP_OFF

Sync    equ    0
In      equ    3
Out     equ    2

        org    0x00

        movwf   OSCCAL           ;calibrating the internal oscillator

        clrf   GPIO

        movlw   B'00111011'
        TRIS   GPIO

        movlw   B'01000010'
        OPTION

        btfsc  GPIO,Sync
        goto   $-1

loop

        clrwdt

        btfss  GPIO,Sync
        goto   $-1

        btfsc  GPIO,In
        bcf   GPIO,Out
        btfss  GPIO,In
        bsf   GPIO,Out

        clrwdt

        btfsc  GPIO,Sync
        goto   $-1

        btfsc  GPIO,In
        bcf   GPIO,Out
        btfss  GPIO,In
        bsf   GPIO,Out

        goto   loop

        end

*****
;
;                               Figure2.ASM
;
;                               This application is made for power nets AC220V 50Hz
;
*****
```

Electromechanical Switch Replacement

```
; Don't use it on 60Hz!  
; It has not back reference to control the current in the triac  
; so it should not be used to drive reactive charges (solenoids etc.),  
; where there is big phase difference between the  
; voltage and the current!  
;  
;*****
```

LIST p=12C508

```
#include "inc\p12c508.inc"
```

```
__config _WDT_ON & _IntrC_OSC & _MCLRE_OFF & _CP_OFF  
  
RAM equ 0x07;Begining of RAM  
  
Sync equ 0  
Btn1 equ 1  
Btn2 equ 3  
Out equ 2  
  
cblock RAM  
    BtnCount1 ;Counters used to delay when buton is pushed  
    BtnCount2  
    Phase ;The value got from the Table  
    Dimmer  
    Flag  
endc  
  
org 0x00  
  
movwf OSCCAL ;calibrating the internal oscillator  
  
goto main
```

```
-----  
; This table makes the dependence  $y=\sin(x)$  linear so, if you  
; use this program to control dimmers and if you increment x by 1  
; up to 63 (0 - 0x3F) and measure the light with luxmeter the dependence  
; will be linear  
;
```

```
Table: andlw .63  
       addwf PCL,F  
       retlw .154  
       retlw .151  
       retlw .146  
       retlw .141  
       retlw .136  
       retlw .132  
       retlw .129  
       retlw .125  
       retlw .122  
       retlw .119  
       retlw .117  
       retlw .114  
       retlw .112  
       retlw .110  
       retlw .108  
       retlw .107  
       retlw .105
```

Electromechanical Switch Replacement

```
retlw      .103
retlw      .102
retlw      .100
retlw      .99
retlw      .97
retlw      .96
retlw      .94
retlw      .93
retlw      .92
retlw      .90
retlw      .89
retlw      .88
retlw      .87
retlw      .85
retlw      .84
retlw      .83
retlw      .82
retlw      .80
retlw      .79
retlw      .78
retlw      .77
retlw      .76
retlw      .74
retlw      .73
retlw      .72
retlw      .71
retlw      .69
retlw      .68
retlw      .67
retlw      .66
retlw      .64
retlw      .63
retlw      .62
retlw      .60
retlw      .58
retlw      .56
retlw      .54
retlw      .52
retlw      .50
retlw      .48
retlw      .45
retlw      .42
retlw      .38
retlw      .34
retlw      .30
retlw      .21
retlw      .0

main
    clrf      GPIO

    movlw     B'00111011'
    TRIS      GPIO

    movlw     B'01000101'    ;clockout/64
    OPTION

;-----
; When 50Hz net is used the period is 20ms and
; the half period is 10ms. 64us X 156 =9.984ms
; the maximum value that the table gives is 154

    movlw     .100
```

Electromechanical Switch Replacement

```
        movwf      BtnCount1      ;Initialize the button counters
        movwf      BtnCount2      ;there will be about 1 sec delay when you
                                   ;push some button. When you hold the putton
                                   ;the output will change within about 0.1 sec

        btfsc      GPIO,Sync
        goto       $-1

loop

        clrwdt

        btfss      GPIO,Sync      ;loops while Sync=0
        goto       $-1

        clrf       TMR0

; First half period

        movf       Dimmer,w
        call       Table          ;converts the value in Dimmer to Phase
        movwf      Phase

Lal

        movf       Phase,w        ;compares the Phase with the timer
        subwf      TMR0,w         ;when the time has come swithes the ouput on
        btfsc      STATUS,C
        goto       Lbl

        bsf        GPIO,Out       ;output on

;Button 1

Btna1

        btfsc      GPIO,Btn1      ;skip if button 1 is pushed else
        goto       Btn1Enda       ;inicializes the BtnCount1

        decfsz     BtnCount1,F     ;if the button was held down for about 1 sec
        goto       BtnEnda        ;(when pushed) or 0.1 sec (after the
        movlw      .10            ;first sec) the value of Dimmer is
        movwf      BtnCount1      ;incremented

        incf       Dimmer,F        ;if the highest value (0x3F) is reached
        btfsc      Dimmer,6       ;no more incrementation is done
        decf       Dimmer,F
        goto       BtnEnda

Btn1Enda

        movlw      .100           ;if the button was not pushed BtnCount is
        movwf      BtnCount1      ;inicialized
        goto       BtnEnda

;the algorithn downwards is like the above

;Button 2

Btna2

        btfsc      GPIO,Btn2      ;skip if button 2 is pushed else
        goto       Btn2Enda       ;inicializes the BtnCount2

        decfsz     BtnCount2,F     ;if the button was held down for about 1 sec
        goto       BtnEnda        ;(when pushed) or 0.1 sec (after the
        movlw      .10            ;first sec) the value of Dimmer is
        movwf      BtnCount2      ;incremented
```

Electromechanical Switch Replacement

```
        decf          Dimmer,F
        btfsc        Dimmer,6
        incf          Dimmer,F
        goto         BtnEnda

Btn2Enda
        movlw        .100
        movwf        BtnCount2

BtnEnda
        clrwdt
        bcf          GPIO,Out

;Second half period
        btfsc        GPIO,Sync      ;loops while Sync=1
        goto         $-1

        clrf         TMR0

        movf         Dimmer,w
        call         Table
        movwf        Phase

Lb1
        movf         Phase,w
        subwf        TMR0,w
        btfsc        STATUS,C
        goto         Lb1

        bsf          GPIO,Out

Btnb1
        btfsc        GPIO,Btn1
        goto         Btn1Endb

        decfsz       BtnCount1,F
        goto         BtnEndb
        movlw        .10
        movwf        BtnCount1

        incf         Dimmer,F
        btfsc        Dimmer,6
        decf         Dimmer,F
        goto         BtnEndb

Btn1Endb
        movlw        .100
        movwf        BtnCount1

Btnb2
        btfsc        GPIO,Btn2
        goto         Btn2Endb

        decfsz       BtnCount2,F
        goto         BtnEndb
        movlw        .10
        movwf        BtnCount2

        decf         Dimmer,F
        btfsc        Dimmer,6
        incf         Dimmer,F
        goto         BtnEndb
```

Electromechanical Switch Replacement

```
Btn2Endb
    movlw    .100
    movwf   BtnCount2

BtnEndb

    bcf     GPIO,Out
    goto   loop

    end

;*****
;
;           Figure3.ASM
;
;   This application is made for power nets AC220V 50Hz
;   Don't use it on 60Hz!
;   It has not back reference to control the current in the triac
;   so it should not be used to drive reactive charges (solenoids etc.),
;   where there is big phase difference between the
;   voltage and the current!
;
;*****

LIST      p=12C671

#include   "inc\p12c671.inc"

__config  _WDT_ON & _IntrC_OSC & _MCLRE_OFF & _CP_OFF

RAM       equ          0x07          ;Begining of RAM

In        equ          0
Sync      equ          3
Out       equ          2

cblock   RAM

        BtnCount1      ;Counters used to delay when buton is pushed
        BtnCount2
        Phase          ;The value got from the Table
        Dimmer
        Flag

    endc

    org     0x00

    call   EndAdd
    movwf OSCCAL      ;calibrating the internal oscillator

    goto  main

;-----
;   This table makes the dependence  $y=\sin(x)$  linear so, if you
;   use this program to control dimmers and if you increment x by 1
;   up to 63 (0 - 0x3F) and measure the light with luxmeter the dependence
;   will be linear
;
```

Electromechanical Switch Replacement

Table:	andlw	.63
	addwf	PCL,F
	retlw	.154
	retlw	.151
	retlw	.146
	retlw	.141
	retlw	.136
	retlw	.132
	retlw	.129
	retlw	.125
	retlw	.122
	retlw	.119
	retlw	.117
	retlw	.114
	retlw	.112
	retlw	.110
	retlw	.108
	retlw	.107
	retlw	.105
	retlw	.103
	retlw	.102
	retlw	.100
	retlw	.99
	retlw	.97
	retlw	.96
	retlw	.94
	retlw	.93
	retlw	.92
	retlw	.90
	retlw	.89
	retlw	.88
	retlw	.87
	retlw	.85
	retlw	.84
	retlw	.83
	retlw	.82
	retlw	.80
	retlw	.79
	retlw	.78
	retlw	.77
	retlw	.76
	retlw	.74
	retlw	.73
	retlw	.72
	retlw	.71
	retlw	.69
	retlw	.68
	retlw	.67
	retlw	.66
	retlw	.64
	retlw	.63
	retlw	.62
	retlw	.60
	retlw	.58
	retlw	.56
	retlw	.54
	retlw	.52
	retlw	.50
	retlw	.48
	retlw	.45
	retlw	.42
	retlw	.38
	retlw	.34
	retlw	.30
	retlw	.21

Electromechanical Switch Replacement

```
        retlw        .0

main

        clrf        GPIO
        clrf        INTCON

        bsf        STATUS,RP0
        movlw       B'00111011'
        movwf      TRIS

        movlw       B'01000101'    ;clockout/64
        movwf      OPTION_REG

        movlw       B'00000110'
        movwf      ADCON1

        bcf        STATUS,RP0

;-----
; When 50Hz net is used the period is 20ms and
; the half period is 10ms. 64us X 156 =9.984ms
; the maximum value that the table gives is 154

        movlw       .100
        movwf      BtnCount1    ;Initialize the button counters
        movwf      BtnCount2    ;there will be about 1 sec delay when you
                                ;push some button. When you hold the putton
                                ;the output will change within about 0.1 sec

        movlw       B'01000001'    ;Inicializes ADC cchannel 0 (GP0)
        movwf      ADCON

        btfsc      GPIO,Sync
        goto       $-1

loop

        clrwdt

        btfss      GPIO,Sync    ;loops while Sync=0
        goto       $-1

; First half period

        clrf        TMR0

        movf        Dimmer,w
        call       Table        ;converts the value in Dimmer to Phase
        movwf      Phase

Lal

        movf        Phase,w      ;compares the Phase with the timer
        subwf      TMR0,w        ;when the time has come swithes the ouput on
        btfsc      STATUS,C
        goto       Lb1

        bsf        GPIO,Out      ;output on

        bsf        ADCON,GO
```

Electromechanical Switch Replacement

```
        btfsc    ADCON,GO
        goto     $-1

        bcf     STATUS,C
        rrf     ADRES,W
        movwf   Dimmer
        bcf     STATUS,C
        rrf     Dimmer,F

        clrwdt
        bcf     GPIO,Out

;Algorithm for second half period is the same as in the first half period

;Second half period

        btfsc   GPIO,Sync    ;loops while Sync=1
        goto   $-1

        clrf    TMR0

        movf    Dimmer,w
        call   Table
        movwf   Phase

Lb1:    movf    Phase,w
        subwf   TMR0,w
        btfsc   STATUS,C
        goto   Lb1

        bsf     GPIO,Out

        bsf     ADCON,GO

        btfsc   ADCON,GO
        goto   $-1

        bcf     STATUS,C
        rrf     ADRES,W
        movwf   Dimmer
        bcf     STATUS,C
        rrf     Dimmer,F

        bcf     GPIO,Out

        goto   loop

EndAdd  IFDEF    __12C671
        org     0x3FF
        ELSE
EndAdd  IFDEF    __12C672
        org     0x7FF
        ENDIF

        end
```

Electromechanical Switch Replacement

NOTES:

Electromechanical Switch Replacement

NOTES:



Electromechanical Switch Replacement

Freezer Protector

*Author: Bret Walters
Inter.tec
Pickering State, Ontario
Canada
email: bretw@ibm.net*

OVERVIEW

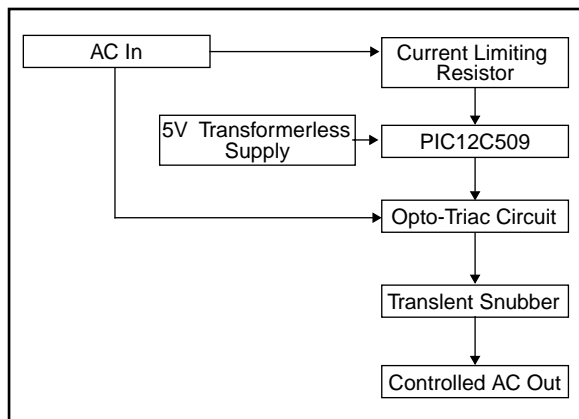
My idea for an electromechanical timer replacement is to use a PIC12C509 to count AC cycles to produce an accurate timer. The delay can be configured in software from 2 to 255 minutes by changing one variable. There is not user interface required. The device simply starts counting AC cycles on power up.

When the time has elapsed, the PIC12C509 triggers the output line which is connected to an optically isolated triac circuit. The PIC12C509 senses the start of the next cycle and powers the triac right on the zero crossing (this reduces wear and tear).

Ideally, this circuit was designed as a freezer protector. It counts 5 minutes from power up and then starts the freezer motor.

During short power failures, restarting the compressor in a freezer just after it has shut off increases wear and tear on the motor. The PIC12C509 prevents this by waiting 5 minutes, then restarting the freezer motor. This gives the refrigerant enough time to settle and therefore reduces load on the freezer motor. Waiting also prevents the high power drain caused by a large motor (the freezer) starting up from occurring at the same time as all the other circuits (if any) that are coming back on line. The reduces load on the power line on startup.

BLOCK DIAGRAM



APPLICATION OPERATION

Hardware

The whole circuit is powered by the AC line. To obtain power for the PIC12C509 and related devices, a transformerless power supply is employed (see Microchip Technical Note TB008 (DS91008)).

The AC cycles are input right from the 117V AC line through a 4.7 Meg resistor which limits the current (see Microchip Application Note AN521 (DS00521)). The GPIO GP1 pin is configured and used as an input.

The circuit waits 5 minutes from power-up and then sets the GPIO GP0 line low. This fires the opto-triac (MOC3023) which in turn fires the triac (Motorola® MAC224A10 or any appropriate triac). The rest of the circuitry on the output provides the correct bias for the triac and opto-triac and includes a “snubbing” circuit. The snubbing circuit reduces inductive voltage transients such as those caused from a motor (such as a freezer motor) connected to this circuit. This improves the noise characteristics and the life of the triac. For more information on this circuit, please see Motorola Application Note AN780 (DS00780).

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.

Electromechanical Switch Replacement

SOFTWARE

The software was designed for modularity. It was believed that the functions created were useful for other projects so it was decided that it would be best to break the functions into subroutines. It would be very easy to remove all of the subroutines and redo the code without them, but due to the nature of this design (speed not critical, 2 level hardware stack available), it is unnecessary.

The functions are discussed here:

- main - the main code

This function controls the calling of all other functions and handles tasks like setting up the I/O lines and setting the output line low at the end of the program.

- Cycle Check

This useful function checks the GPIO GP1 pin continuously. By watching for a high input and then a low input, the rising edge of the AC input is found. Plus, due to the speed of the PICmicro, this is the zero crossing as well. After this point is found, the function RETURNS control to main.

- Initialize Cycle Timer

This function works with the function "Cycle Timer". Calling it initializes all variables used in the "Cycle Timer" function to their original values. The values are set in the beginning of the code as a series of variables (using EQUates). The variables are `cti1`, `cti2` and `cti3`. They are the 1/60 seconds, seconds and minutes initialization values respectively. "cti" stands for Count Initialize. The values initialize CTR1, CTR2 and CTR3 respectively, these are used for the function "Cycle Timer".

- Cycle Timer

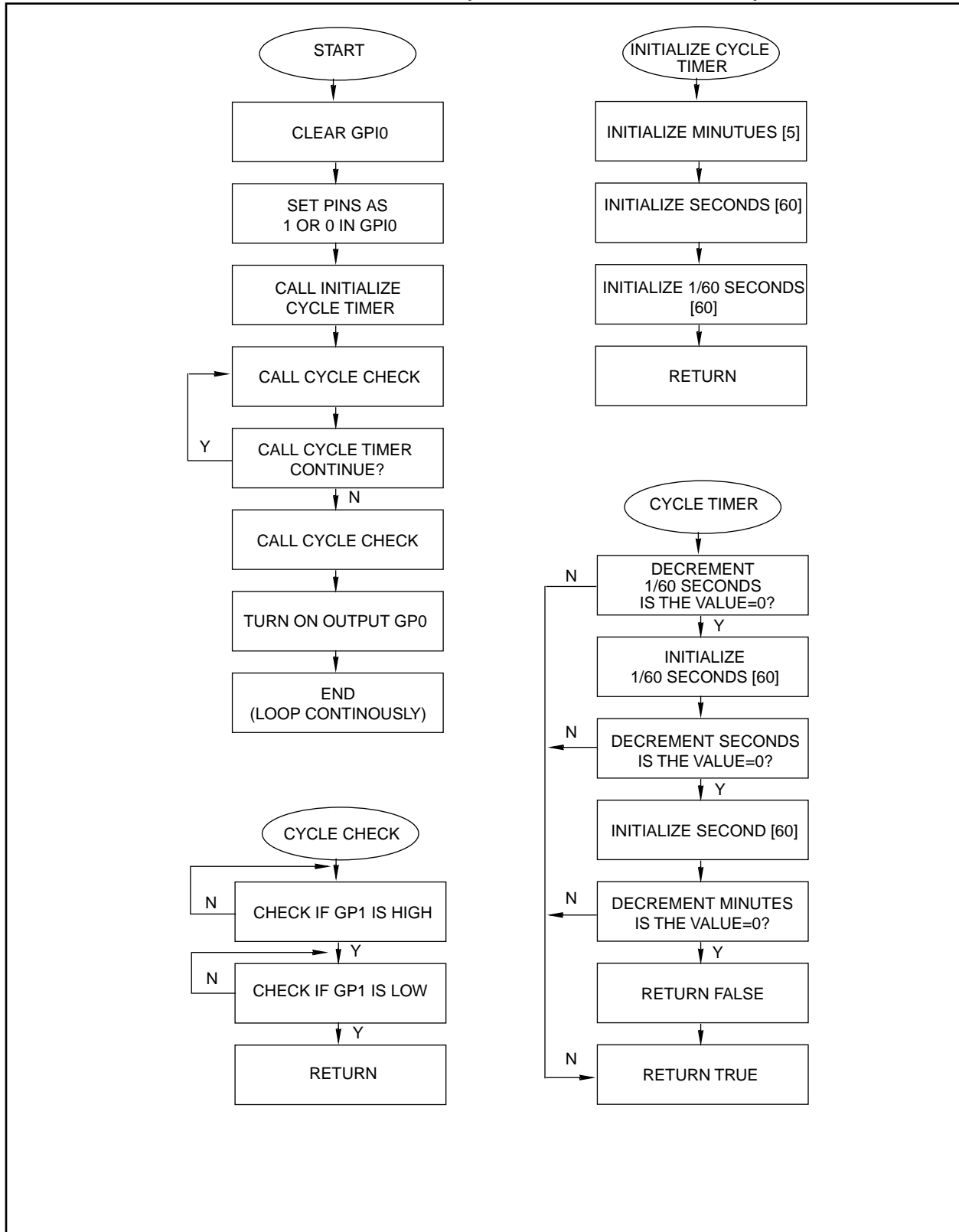
This function is intended for use with the "Cycle Check" function but can be used anywhere. The function decrements 3 registers (CTR1, CTR2 and CTR3) in a count down fashion. It counts once updating the appropriate registers, then exits. In this submission the registers count 1/60s of a second, seconds and minutes.

On exit, the function passes a value to the W register appropriate as to if the count is finished or not. The count is finished when the "minutes" counter reaches 0. If the count is not finished, a 1 is returned, signifying that the counter needs to continue. If the count is finished, a 0 is returned. The main code handles these conditions.

The counting is accomplished by decrementing and looking for a 0 result. If a 0 is found, the 0 is reinitialized to the correct value and then the next variable is decremented. See the code and/or the Flow Chart for more details on this function.

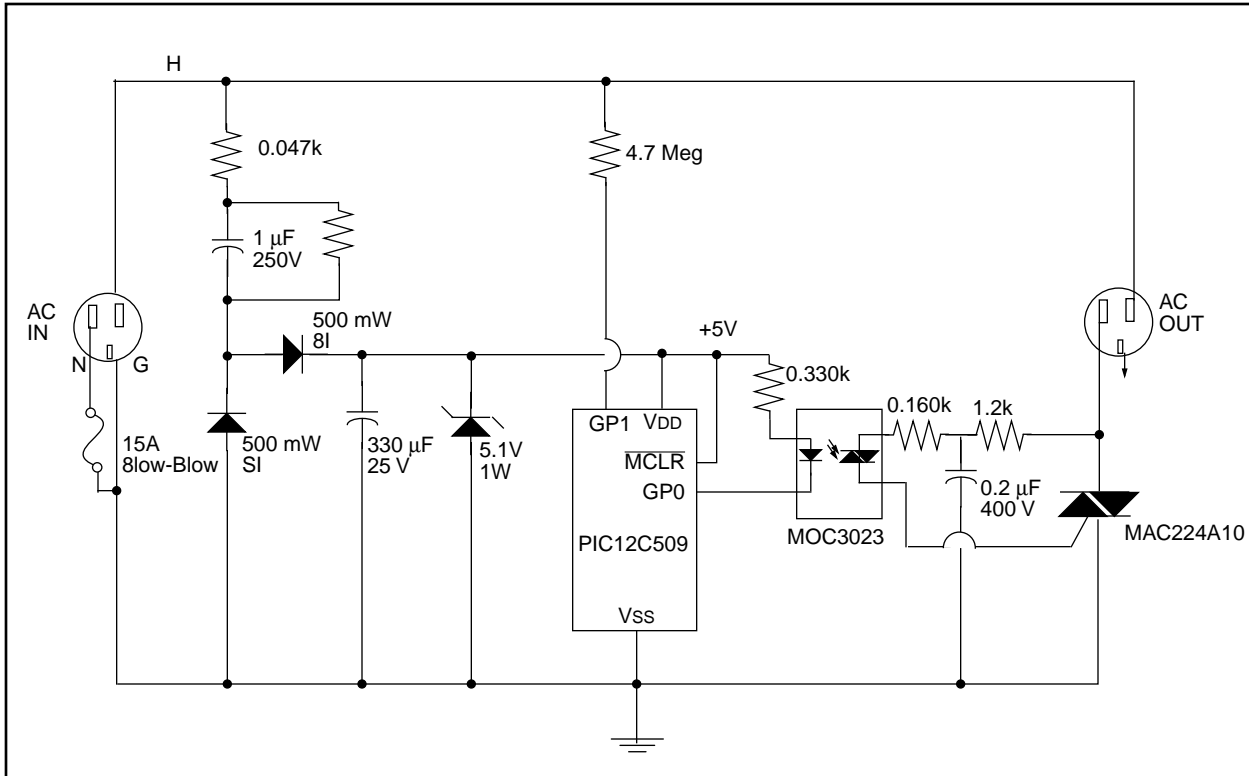
Electromechanical Switch Replacement

FLOW CHART FOR AC TIMER SWITCH (FREEZER PROTECTION)



Electromechanical Switch Replacement

GRAPHICAL HARDWARE REPRESENTATION



MICROCHIP HARDWARE DEVELOPMENT TOOLS USED

Initial code was tested on the PIC16C84 due to its convenient EEPROM configuration. Later versions were developed on the PIC12C509 - EPROM version. All parts were programmed using the PICSTART[®] Plus Development Programmer, part number: 10-00157 by Microchip Technology Inc. This was obtained through the complete PICSTART Plus Development System obtained from the 1997 Toronto conference.

Assembler/Compiler Version

MPLAB[™] for Windows[®]/16 version 3.22.00 Microchip Technology Inc., running on Windows 95. Processor version 6.2, Disassembler version 2.0. Copyright Microchip Technology Inc. 1995. This was also obtained through the PICSTART Plus 1997 Toronto Conference.

Electromechanical Switch Replacement

APPENDIX A: SOURCE CODE

```
;------INTRODUCTION-----
;This code is by Bret Walters 1997. Version 0.9
;This is an unpublished work.
;
;This program is developed for the PIC12C509 design contest session
;2. It uses a 12C508/09 to protect an electrical device.
;The protection is that as an AC power failure occurs, all
;devices attempt to turn on at once, causing havoc with the AC
;line.
;
;Devices like freezers and the like could be turned on
;significantly after the restoring of power. This example
;uses a 5 minute delay.
;
;The following code was designed for the 12C509 (the part I
;had on hand) and has been successfully simulated using
;MPLAB 3.22.00

;------FUNCTION-----
;This code counts the number of AC cycles which are 1/60 s
;apart (or 1/50 s in Europe - a simple code change), via
;GP1. Pulses enter via the AC line through a 4.7 Meg resistor
;to the PIC12C509 (at GP2). Output from the PIC12C509 at GP0 drives an
;optocoupler which drives a triac that turns on the electrical
;device. A solid state relay could also replace the
;triac/optocoupler combination.
;
;Zero crossings (in the rising edge) are counted providing an
;accurate timing source. This is done by looking for a low
;then looking for a high input. One of the best features of
;this design is that the device is only turned on at the zero
;crossing, reducing noise and eliminating the need for a zero
;crossing circuit

;------ACKNOWLEDEMENTS-----
;The power supply used is featured
;in the Microchip Application Notes. The Triac/Optocoupler
;configuration is a design of Motorola Inc.

;------FUTURE ENHANCEMENTS-----
;Using careful timing would allow the PIC12C509 to turn on the
;triac at different points on the AC cycle
;(since the zero crossing point is known) to provide a
;dimmer for other circuits.

;------PART-----
list p=12c509
include "p12c509.inc"

;------VARS-----
ScratchPadRam equ 0x07
CTR1 equ ScratchPadRam+0
CTR2 equ ScratchPadRam+1
CTR3 equ ScratchPadRam+2
CHECK1 equ ScratchPadRam+3
w equ .0
f equ .1
z equ .2
PORTA equ 0x06
STATUS equ 0x03
cti1 equ .60
cti2 equ .60
cti3 equ .5
FALSE equ .0
TRUE equ .1
```

Electromechanical Switch Replacement

```
;------CODE-----
      org      0          ;start address 0
      clr     PORTA      ;clear output to prevent error and noise on powerup
      bsf     PORTA,0    ;set appliance to off state
movlw  0x1E
      tris    PORTA      ; as inputs except GP0
      call   icytmr      ;call initialize cycle timer for cyctmr routine
chkagn call   cycchk      ;call cycle check, look for 1 AC cycle and return
      call   cyctmr      ;call cycle timer, count to 5 minutes and exit
movwf  CHECK1           ;load exit value for testing
movf   CHECK1,f        ; set flags appropriate for CHECK1
      btfsz  STATUS,z    ;a 0 returned from cyctmr will end the counting
      goto   chkagn      ;check again for next cycle if count not finished
      call   cycchk      ; call cycle check, after 1 AC cycle,
      bcf    PORTA,0     ;turn on appliance
stphre goto   stphre    ;stop here

;***CYCLE CHECK - Get rising edge of AC cycle on GP1***
cycchk  btfsz   PORTA,1   ;check AC input at GP1 for 0
        goto   cycchk    ;still in high cycle check again
low_ok  btfsz   PORTA,1   ;check AC input at GP1 for 1
        goto   low_ok    ;still in low cycle check again
        return           ;found zero crossing - rising edge

;***INITIALIZE CYCLE TIMER - Initializes cyctmr***
icytmr  movlw   cti3      ;initialize
        movwf  CTR3      ; counter 3 (counts minutes)
        movlw  cti2      ;initialize
        movwf  CTR2      ; counter 2 (counts seconds)
        movlw  cti1      ;initialize
        movwf  CTR1      ; counter 1 (counts 1/60ths of a second)
        return

;***CYCLE TIMER - counts calls since initialization to create a specific delay time***
cyctmr  decfsz  CTR1,f    ;nest 1:Decrements preloaded value
        ;      [1s (60Cy) for nest 1]
        retlw  TRUE      ; to 0 and then,
        movlw  cti1      ;initializes
        movwf  CTR1      ; counter 1 (1/60 seconds)
        decfsz CTR2,f    ;nest 2: Decrements preloaded value
        ;      [1min for nest 2 and 1]
        retlw  TRUE      ; to 0 and then,
        movlw  cti2      ;initializes
        movwf  CTR2      ; counter 2 (seconds)
        decfsz CTR3,f    ;nest 3: Decrements preloaded value
        ;      [5min for nest 3 to 1]
        retlw  TRUE      ; to 0 and then,
        retlw  FALSE     ; exits (0)
end
```



Electromechanical Switch Replacement

Implementing Software Timer Interrupts on PICmicro™ MCUs without Hardware Interrupt

*Author: Marc Hoffknecht
Aachen, Germany*

APPLICATION OPERATION

This code is very short (8 bytes) and efficient (max. 10 cycles including call and return) and, therefore, is ideal for all applications taking advantage of the high execution speed of the PICmicro™ microcontrollers (MCU).

This code example implements a software timer-interrupt on PICmicro MCUs which do not feature hardware interrupts. That way, the user can implement timers with very long periods or can debounce buttons or anything.

The idea is that the user calls the routine 'Handle Interrupt' every now and then, which will check the on-chip timer to see whether an overflow occurred. If so, this routine will call the users-interrupt routine.

Detailed description how to use it:

1. Set up the on-chip timer and prescaler that way, that its overflow-rate equals the time you need your interrupt service routine called.
2. Write your interrupt service routine.
3. Execute 'CALL Handle Interrupt' every now and then.

See example code on Page 2.

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.

Electromechanical Switch Replacement

EXAMPLE SOURCE CODE

EPROM usage: 8 byte
RAM usage: 1 byte
Clock Cycles including CALL and RETLW: 9 cycles if no interrupt pending, otherwise 10 + length of users interrupt-routine

Code:

```
InterruptADDWF OldTMR0
; users timer-interrupt code here
    RETLW 0

;

CBLOCK
OldTMR0; only variable
ENDC
```

```
HandleInterrupt MOVF OldTMR0, W; check for TMR0-overflow
    SUBWF TMR0, W ; overflow, if OldTMR0 > TMR0
    BTFSS carry
    GOTO Interrupt
    ADDWF OldTMR0
    RETLW 0
```

Example code:

```
InterruptADDWF OldTMR0
    DECF MyTimer; users timer-interrupt code:
RETLW 0 ; decrease 'MyTimer'
; every 4.096ms @ 4MHz
;

CBLOCK
OldTMR0 ; only variable
ENDC
```

```
HandleInterrupt MOVF OldTMR0, W; check for TMR0-overflow
    SUBWF TMR0, W ; overflow, if OldTMR0 > TMR0
    BTFSS carry
    GOTO Interrupt
    ADDWF OldTMR0
    RETLW 0
```

;

```
Main    MOVLW b'xx00011'; on-chip timer will overflow every
        OPTION ; lus * 16 (prescaler) * 256 = 4.096ms

        ; some code
```

```
        CALL HandleInterrupt; do this if you don't need the
; contents of w and status saved,
; at least once every 4.096ms !
; Call it often to increase accuracy !
        ; some code
```

Electromechanical Switch Replacement

NOTES:



Light Switch with Delay Turn-Off

Author: *Brian lehl*
brian@dls.net

OVERVIEW

For cost effectiveness a PIC12C508 microcontroller is used. This light switch allows the user to turn off a light switch then have the circuit wait until the user has left the room to actually turn the light off. This increases safety in the home or work place by allowing people to not have to walk through the dark. There is no delay when the light is turned on. This device is intended to be used in stand alone lamps or in wall light switches. Therefore, this device needs to be low cost, small in size, and powered off of the AC power line.

APPLICATION OPERATION

This circuit monitors the state of a light switch. If the switch turns on, the microcontroller turns on a triac which in turn switches on the light. When the switch is turned off, the microcontroller waits 15 seconds (or other predetermined time), then turns the triac off which in turn switches the light off.

Since the PIC12C508 draws so little current, a very simple, low cost power supply circuit is used which does not need a transformer. This allows the entire circuit to be very small. The PIC12C508 is perfectly suited to this application because it doesn't need any supporting circuitry. Also, this device is in an amazingly small package which will allow the circuit to be mounted in a standard wall light switch. Even though the program memory in the PIC12C508 is limited to 512 bytes, that is still about 10 times more than is required for this simple application. This allows the microcontroller to be inexpensive enough to be used in cost sensitive applications like this one where it replaces standard discrete devices.

MICROCHIP TOOLS USED

Assembler/Compiler version:

MPLAB™ 3.22.00

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 the author.

Electromechanical Switch Designs

APPENDIX A: SOURCE CODE

```
; Title "Light Switch"
; Subtitle"Version 1.0"
; Written by Brian Iehl 6/28/97

list p=12C508

INCLUDE c:\apps\mplab\p12c508.inc
SetIO equ B'00001000' ; 0 for output, 1 for input

GPIO0 equ 0
GPIO1 equ 1
GPIO2 equ 2
GPIO3 equ 3
GPIO4 equ 4
GPIO5 equ 5

SwState equ GPIO3 ; Input
SwOn equ 0 ; Switch closed GPIO0 is Low
SwOff equ 1 ; Switch open GPIO0 is high
SwValue equ B'00001000' ; Used to test GPIO3 bit

LitCntl equ GPIO0 ; Output
LitOn equ 1 ; Hi to turn light on
LitOff equ 0 ; Lo to turn light off

Debounce equ D'100' ; mS to wait for switch to settle
LitDelay equ D'15' ; S to wait for light to turn off

ScratchPadRam equ 0x07
OldSwState equ ScratchPadRam+0 ; Save last state
NewSwState equ ScratchPadRam+1 ; Save current state
DelayValue equ ScratchPadRam+2 ; For DelayRoutine
SDelayValue equ ScratchPadRam+3 ; For SDelayRoutine

;***** Macros *****

MOVLF MACRO LL, FF ; Move Literal to register file
MOVLF LL ; Load literal
MOVWF FF ; Store in register file
ENDM ; end MOVLF

mSDelay MACRO mS ; Assumes 4 MHz clock; Needs global DelayValue
; Number of mS to delay up to 255 mS
; each clock cycle is 1 uS = .001 mS

LOCAL Loop, SetTmr
MOVLF mS, DelayValue ; store number of mS delay
CLRWDTC ; avoid unitentional reset

SetTmr MOVLF B'00000111' ; Set prescaler to 256, clear PSA, Clear T0CS
OPTION ; store prescaler value

Loop MOVLF -4, TMR0 ; 4 * 256 = 1024 uS ~ 1 mS
MOVF TMR0, w ; force check zero
BTFSZ STATUS, Z ; w = 0 if same, so Z is set
goto Loop ; not 0 so loop again
; one more mS passed
DECFSZ DelayValue, f ; count down number of mS
goto SetTmr ; not done reset timer
; if DelayValue = 0 then done
ENDM ; end mSDelay

SDelay MACRO S ; Macro SDelay; Needs global SDelayValue
; Number of Seconds delay up to 63
LOCAL Loop
```

Electromechanical Switch Designs

```

                                MOVLF  S*4,      SDelayValue ; store number of S delay
Loop      mSDelay D'250'          ; Delay 0.25 sec
                                DECFSZ  SDelayValue,f      ; count down number of S
                                goto    Loop                ; not done reset timer
                                                    ; if DelayValue = 0 then done

                                ENDM                        ; end SDelay

;*****

                                org     0x0A              ;start address 0x0A
                                goto    Start
                                org     0x10

;
Start

Setup      MOVLF  SetIO           ; Load IO configuration byte
                                TRIS   GPIO              ; Set GPIO with contents of w
                                CLRF   OldSwState        ; Init RAM
                                CLRF   NewSwState
                                CLRF   DelayValue
                                CLRF   SDelayValue

                                                    ; get state at power up
                                MOVF   GPIO,    w        ; read GPIO register
                                ANDLW  SwValue          ; Clear all bits except SwState
                                BTFSS  STATUS,  Z        ; if SwState = Lo
                                goto    TurnOff

TurnOn     BSF    GPIO,    LitCntl  ; Turn Light on
                                MOVLF  SwOn,    OldSwState ; Save lastest switch state
                                goto    ReadIt

TurnOff    BCF    GPIO,    LitCntl  ; Turn Light Off
                                MOVLF  SwOff,   OldSwState ; Save lastest switch state

ReadIt     mSDelay Debounce        ; Switch Debounce
                                MOVF   GPIO,    w        ; read GPIO register
                                ANDWF  SwState,  w      ; Clear all bits except SwState
                                MOVWF  NewSwState      ; save for later reference
                                XORWF  OldSwState,w    ; Are they the same
                                BTFSC  STATUS,      Z   ; w = 0 if same, so Z is set
                                goto    ReadIt         ; same to loop again
                                                    ; not same so state was changed
                                MOVF   NewSwState,w    ; Load new switch state
                                BTFSC  STATUS,      Z   ; w = 0 if SwOn, so Z is set
                                goto    TurnOn         ; Turn light on

SetTim     SDelay LitDelay         ; Wait before turning light off
                                goto    TurnOff        ; Turn light off

                                END
```



Electromechanical Switch Replacement

NETSWITCH

*Author: Csaba Bardos
RASS Ltd.
Tiszafured, Hungary
email: cbardos@mail.inext.hu*

APPLICATION OPERATION

This gadget is designed for individually dimming lamps, turning on/off electrical devices, etc. It accepts commands via serial lines individually, (i.e., more than one NETSWITCH can connect to same serial line, or you can use its switches to control it).

You can handle up to 256 turning points with this device, and you can control them from one central station. (The central station description is not included in this text). There are two modes on device: Manual or Serial.

NODE-ADDRESSBYTE-COMMAND-COMMANDPARAM-ENDBL

Switching between modes is only allowed from serial commands. It starts Manual Mode when powered up.

Manual Mode

There are three buttons on the device. One for turning on/off the load, the other two are push buttons which let you dim the load. When the on/off switch is turned on, dimming is not allowed. When turned off, dimming is allowed and you have to dim to zero intensity if you want to turn off.

Serial Mode

The device is able to receive (only) data from the serial line at 1200 bps. Being connected to the same serial line and to more than one NETSWITCH is allowed, because every NETSWITCH has a unique number to identify. (This number not changeable, it is set compile time, though, this is not benefit.)

The NETSWITCH accepts blocks of bytes only as follows:

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.

Electromechanical Switch Replacement

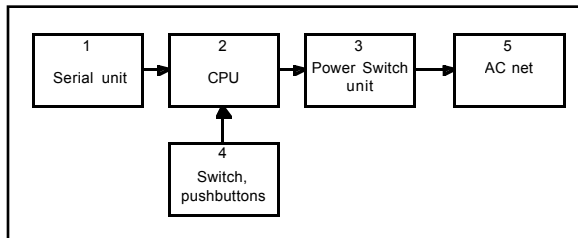
Where:

- NODE: NODE command.
- ADDRESSBYTE: This is a unique number.
- COMMAND: One of commands below.
- COMMANDPARAM: Optional, it depends from command.
- ENDBL: This is a terminator command.

The acceptable commands are:

- NODE: This command means the start of block, its value is ASCII 'N'.
- MANU: This command puts the NETSWITCH to manual mode. When in this mode, it accepts its switches. Its value is ASCII 'M'.
- SERI: This command puts the NETSWITCH to serial mode. When in this mode, it accepts serial commands only. Its value is 'S'.
- SWOFF: This command turns off the NETSWITCH. Its value is ASCII 'F'.
- SWON: This command turns on the NETSWITCH. Its value is ASCII 'O'.
- DIMM: This command dims the load, according to its parameter. Its value is ASCII 'D'.
- ENDBL: This is END of BLock command, every block has to end with it. Its value is ASCII 'E'.

BLOCK DIAGRAM



MICROCHIP HARDWARE DEVELOPMENT TOOLS USED:

Assembler/Compiler Version:

MPASM v1.40

Electromechanical Switch Replacement

APPLICATION OPERATION

Hardware Description

It consists of three main parts:

1. The power supply. The gadget obtains it from another 5V source now (e.g., it comes with serial line).
2. The PIC12C508. It is controlled in two ways.
 - a) In manual mode with push buttons and switch.
 - b) In serial mode with serial commands.

The PIC12C508 runs with its on-chip (internal) RC oscillator, nominally 4 MHz. The PIC12C508 controls the power switching unit by a PWM signal. It is available on pin 7 of PIC12C508.

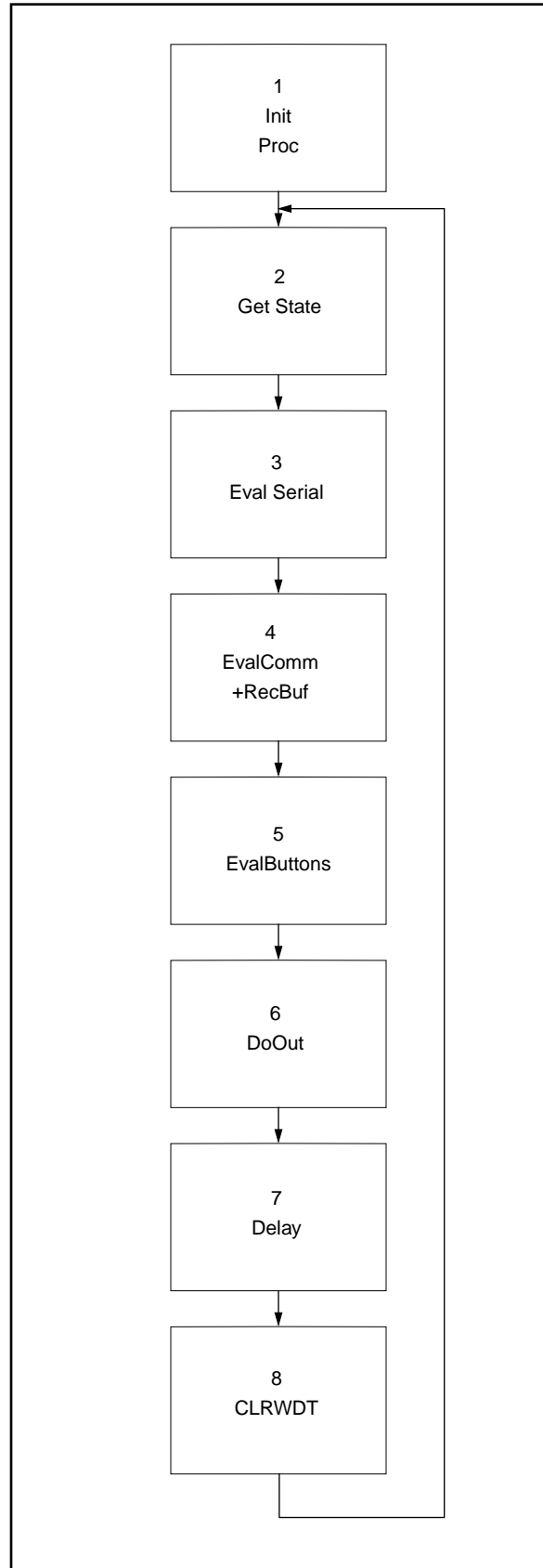
3. The power switching unit. This unit switches (or dims) the load. For a half period:
 - a) When the period starts, the zener stabilizes voltage at 10V.
 - b) The opto-coupler receiving side transistor, NPN, and PNP transistor are voltage controlled current source. This current is available on collector of PNP. It charges a capacitor. When this capacitor reaches threshold of the UJT (uni-junction transistor), it discharges the cap and produces an ignition pulse for thyristor (SCR). You can choose a right thyristor for your requirements.

Software Description

The software contains a loop which gets the state of input pins, then evaluates serial communication, the commands received via the serial line, the state of buttons and then does the appropriate action, i.e. controls SCRs with a PWM sign.

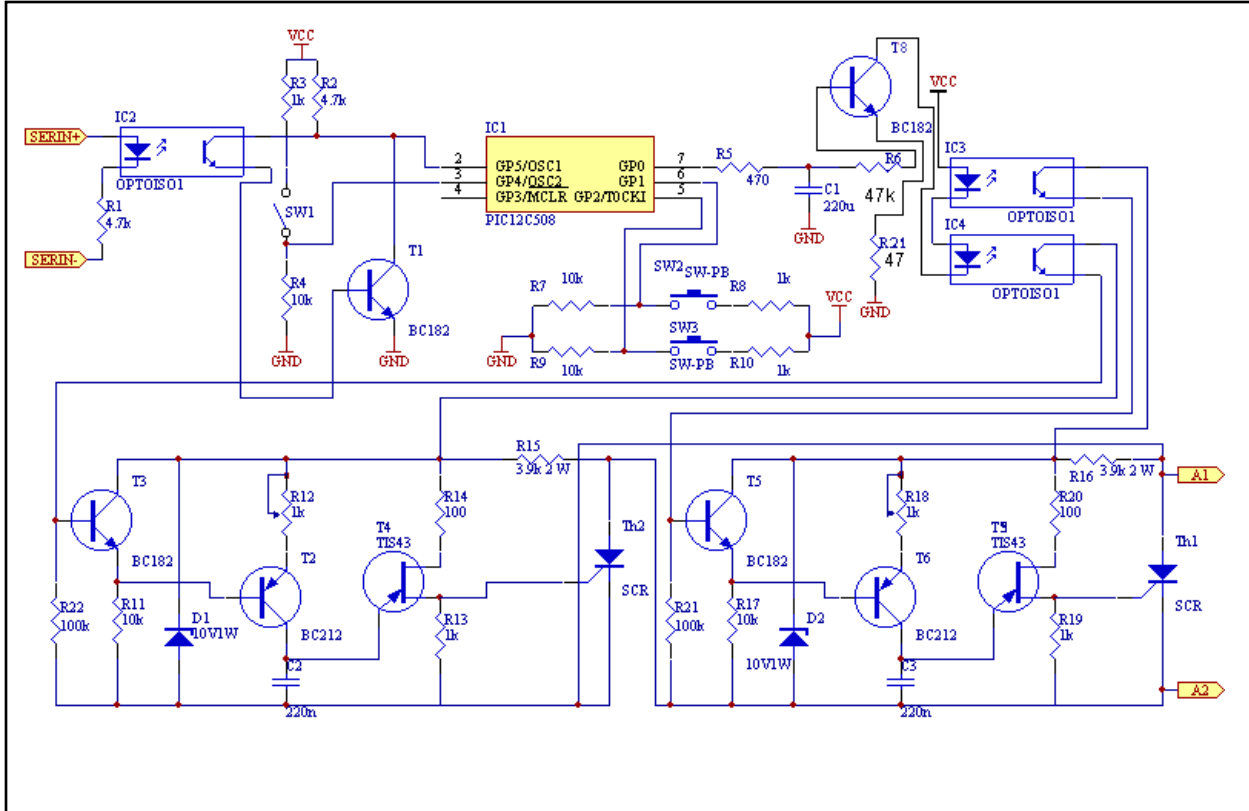
There is a delay subroutine in the loop, which controls the time usage. When the device is not receiving a serial sign, the delay routine does a half time cycle (half of one bit time of 1200 bps). When the device is receiving, it does a complete time cycle (one bit time of 1200 bps). This routine is based on the TMR0. (TMR0 is in timer mode, counts internal clock and has a prescaler 1:4). For additional information, see the file `netsw.asm`, it has many comments.

FLOW CHART



Electromechanical Switch Replacement

GRAPHICAL HARDWARE REPRESENTATION



Electromechanical Switch Replacement

APPENDIX A: SOFTWARE LISTING

```
title "NETSW.ASM V1.0 "  
  
; Electromechanical switch replacement.  
; This program controls the NETSWITCH.  
  
        list p=12c508  
  
#include "p12c508.inc"  
  
; The configuration bits are configured as follows:  
; MCLRE-disabled (0)  
; CP-disabled (1)  
; WDTE-enabled (1)  
; OSC-internal RC (10)  
  
; The next definitions are the commands that accepted by NETSWITCH.  
  
#define NODE    0x4E                ; NODE address, ASCII "N"  
#define MANU    0x4D                ; MANUal mode, ASCII "M"  
#define SERI    0x53                ; SERIal mode, ASCII "S"  
#define SWOFF   0x46                ; Switch turn OFF, ASCII "F"  
#define SWON    0x4F                ; Switch turn ON, ASCII "O"  
#define DIMM    0x44                ; DIMMing, ASCII "D"  
#define ENDBL   0x45                ; END of BLock, ASCII "E"  
  
; Each block that controls the NETSWITCH has to meet the next form:  
; NODE-ADDRESSBYTE-COMMAND-COMMANDPARAM-ENDBL  
; NODE - Command NODE  
; ADDRESSBYTE - This is the NODE address  
; COMMAND - COMMAND byte, see above (SERI, MANU, SWOFF, SWON, DIMM)  
; COMMANDPARAM - One byte, for DIMM command only  
;                                     (e.g. DIMM 0x7F means half intensity)  
; ENDBL - END of BLock  
  
NETSWOPTION    equ 0xC1            ; Bin: 11000001, Tim. presc.to TMR0, 1:4  
NULL           equ 0x00  
EIGHT         equ 0x08  
GPSET         equ 0xFE            ; Only Bit 0 is output  
BLSIZE        equ 0x05            ; BLockSIZE, it is 5 now. See above.  
MYADDR        equ 0x10            ; <- This is an example  
MINPWM        equ 0x00            ; MINimal PWM value, i.e. totally turn off  
MAXPWM        equ 0x01            ; MAXimal PWM value  
BUTTONCYCLE   equ 0x7F            ; For button delaying  
HALFTIME      equ 0x92            ; This is a half cycle value. (416 usec)  
  
ReadBuf       equ 0x08            ; Holds input info  
SWStatus      equ 0x09            ; Holds all system status  
NumBits       equ 0x0A            ; Number of bits  
Counter       equ 0x0B            ;  
EvalComBuf    equ 0x0C            ; Temporary byte for holding changes  
PWMCycleCount equ 0x0D            ; For PWM signal  
PWMPulseCount equ 0x0E            ; For PWM signal  
RecBuf        equ 0x0F            ; Receive Buffer  
DIMMParam     equ 0x10            ; DIMming Parameter  
ButtonCounter equ 0x11            ; Button delaying  
BLTABLE       equ 0x14            ; BLock TABLE starts here  
  
; The next bits are used in SWStatus byte  
  
RecInPr       equ 0x00  
NewByte       equ 0x01  
BLInPr        equ 0x02  
ManMode       equ 0x03  
Pulse         equ 0x04
```

Electromechanical Switch Replacement

```
NewBL          equ 0x05
PWMUpdate      equ 0x06

; The next bits are used in GPIO

Rx             equ 0x05
UpButt        equ 0x02
DownButt      equ 0x03
SW            equ 0x04
PWMOut        equ 0x00

MSB            equ 0x07

org 0x00

Start          call Init          ; Call Init procedure
MainLoop      call GetState       ; Get all input state
              call EvalSerial    ; Evaluate serial input
              call RecBlock      ; Receive all byte in block
              call EvalCommand   ; Evaluate command via serial input
              call EvalButtons   ; Evaluate pushbuttons and switch
              call DoOut         ; Do appropriate action
              call Delay         ; Delay cycle
              clrwdt             ; Clear watchdog
              goto MainLoop      ; Do again

Init          movlw NETSWOPTION   ;
              option            ;
              movlw EIGHT+1     ;
              movwf FSR         ;
              movlw 0x0D        ;
              movwf 0x08       ;
DoAgainClear  decfsz 0x08,F      ;
              goto ClearOneRAM  ;
Continue      movlw EIGHT        ;
              movwf NumBits     ;
              movlw GPSET       ;
              tris GPIO         ;
              movlw HALFTIME    ;
              movwf TMR0        ;
              retlw NULL        ;
ClearOneRAM   clrf INDF          ;
              incf FSR,F        ;
              goto DoAgainClear ;
GetState      movf GPIO,W        ; Read pins
              movwf ReadBuf     ; Put to ReadBuf
              retlw NULL        ; And return

EvalSerial    btfsc SWStatus,RecInPr ; RECeiving IN PProgress bit test
              goto NextBit      ; Set, the next bit comes
              btfss ReadBuf,Rx   ; RX bit test, start condition true ?
              bsf SWStatus,RecInPr ; Yes, set RECeiving IN PProgress bit
              retlw NULL        ; No, return
NextBit       btfsc ReadBuf,Rx   ; RX bit test
              bsf RecBuf,MSB     ; If RX set, set MSB of RecBuf
              bcf STATUS,C       ; Clear Carry
              rrf RecBuf,F       ; Rotate Right
              decfsz NumBits,F   ; Is there more bits ?
              retlw NULL        ; Yes, return
              bcf SWStatus,RecInPr ; No, clear status bit
              bsf SWStatus,NewByte ; This is new byte
              movlw EIGHT        ; Prepare next serial receiving
              movwf NumBits     ;
              retlw NULL        ; Return

; This routine called RecBlock waits until all element of the command
; block is received by setting BLInPr (BBlock receiving In PProgress) bit in
```

Electromechanical Switch Replacement

; SWStatus. It write the received bytes to BLTable.

```
RecBlock          btfss SWStatus,NewByte          ; Is there new byte ?
                  retlw NULL                      ; No, return
                  bcf SWStatus,NewByte           ; Clear new byte status
                  btfsc SWStatus,BLInPr         ; Is receiving in progress ?
                  goto NextByte                 ; Yes, go to NextByte
                  bsf SWStatus,BLInPr          ; No, sets BLInPr bit
                  movlw BLTABLE                ; BLTABLE to
                  movwf FSR                     ; FSR
                  movlw BLSIZE                 ; BLSIZE to
                  movwf Counter                ; Counter
NextByte          movf RecBuf,W                  ; Put contents to
                  movwf INDF                    ; (FSR), i.e. to BLTable
                  incf INDF,F                  ; Points to next element
                  decfsz Counter,F             ; Is there more byte ?
                  retlw NULL                    ; Yes, return
                  bcf SWStatus,BLInPr         ; No, clears BLInPr bit
                  bsf SWStatus,NewBL          ; This is new block
                  retlw NULL                    ; Return, the BLTable is ready to use
```

; The routine called EvalCommand evaluates the BLTable.

```
EvalCommand       btfss SWStatus,NewBL          ; Is there new block ?
                  retlw NULL                    ; No
                  btfsc SWStatus,BLInPr       ; Is block receiving in progress ?
                  retlw NULL                    ; Yes, return
                  movlw BLTABLE; No, BLTABLE to
                  movwf FSR                     ; FSR
                  movlw NODE                   ; This must be the first command
                  call EvalCell                 ; Is the command righth ?
                  btfss STATUS,Z               ;
                  goto ClearTable              ; No, go to ClearTable
                  movlw MYADDR                 ; Yes,
                  call EvalCell                 ; Check, NODE ADDRESS is matching ?
                  btfss STATUS,Z               ;
                  goto ClearTable              ; No, go to ClearTable
                  movlw MANU                   ; Yes,
                  call EvalCell                 ; Is the next command MANU ?
                  btfss STATUS,Z               ;
                  goto ChkSERI                  ; No, go to ChkSERI
                  bsf EvalComBuf,ManMode; Yes, sets ManMode
                  goto ChkEND                   ; Go to ChkEND
ChkSERI           movlw SERI                    ;
                  call EvalCell                 ; Is the command SERI ?
                  btfss STATUS,Z               ;
                  goto ChkSWON                 ;
                  bcf EvalComBuf,ManMode; Yes, clears ManMode
                  goto ChkEND                   ;
ChkSWON           movlw SWON                    ;
                  call EvalCell                 ; Is the command SWON ?
                  btfss STATUS,Z               ;
                  goto ChkSWOFF;
                  movlw MAXPWM;
                  movwf DIMMParm; DIMMParm sets to MAXPWM
                  goto ChkEND                   ;
ChkSWOFF          movlw SWOFF                   ;
                  call EvalCell                 ; Is the command SWOFF ?
                  btfss STATUS,Z               ;
                  goto ChkDIMM                 ;
                  movlw MINPWM                 ;
                  movwf DIMMParm; DIMMParm sets to MinPWM
                  goto ChkEND                   ;
ChkDIMM           movlw DIMM                    ;
                  call EvalCell                 ; Is the command DIMM ?
                  btfss STATUS,Z               ;
```

Electromechanical Switch Replacement

```

                                goto ChkEND                ;
ChkParam                        movf INDF,W                ; Yes, the Parambyte is
                                movwf DIMMParam           ; transferred to DIMMParam
                                incf FSR,F               ;
ChkEND                          movlw ENDBL              ; Is the command ENDBL ?
                                call EvalCell            ;
                                btfss STATUS,Z           ;
                                goto ClearTable          ; No, go to ClearTable
                                movf EvalComBuf,W         ; Yes, all changes are written to
                                iorwf SWStatus,F         ; SWStatus
ClearTable                      movlw BLTABLE            ; BLTABLE to
                                movwf FSR                ; FSR
                                movlw BLSIZE             ; BLSIZE to
                                movwf Counter           ; Counter
ClearCycle                      clrf INDF                ; Clears one element
                                incf FSR,F               ; Points to next element
                                decfsz Counter,F; Is there more byte ?
                                goto ClearCycle          ; Yes,
                                movlw NULL              ; No, all changes are discarded
                                movwf DIMMParam;
                                bcf SWStatus,NewBL;
EvalCell                        retlw NULL              ; Return
                                subwf INDF,W             ; W and INDF are equal ? ( affects Z)
                                btfss STATUS,Z           ;
                                retlw NULL              ; No, (Z remains)
                                incf FSR,F               ; Yes, points next element
                                movlw NULL              ; Sets Z, since NULL = 0
                                retlw NULL              ; return (Z remains)

```

; The routine called EvalButtons evaluates status of the buttons, i.e.
; it sets DIMMParam according to the status of buttons. It works manual
; mode (MANU bit set) of course.

```

EvalButtons                    btfss SWStatus,ManMode; Are buttons enabled ?
                                retlw NULL              ; No
                                btfss ReadBuf,SW        ; Yes, Does Switch turn on ?
                                goto ChkUpButt         ; No, go to check pushbuttons
SWTurnOn                       movlw MAXPWM            ; Yes, turn on
                                movwf DIMMParam;
                                retlw NULL              ;
ChkUpButt                      btfss SWStatus,UpButt   ;
                                goto ChkDownButt       ;
                                decfsz ButtonCounter,F;
                                retlw NULL              ;
                                movlw MAXPWM            ;
                                subwf DIMMParam,W       ;
                                btfss STATUS,Z           ;
                                decf DIMMParam,F        ;
ChkDownButt                    btfss SWStatus,DownButt ;
                                goto RetFromButt       ;
                                decfsz ButtonCounter,F;
                                retlw NULL              ;
                                movlw MINPWM            ;
                                subwf DIMMParam,W       ;
                                btfss STATUS,Z           ;
                                incf DIMMParam,F        ;
RetFromButt                    movlw BUTTONCYCLE       ;
                                movwf ButtonCounter    ;
                                retlw NULL              ;

```

; The routine called DoOut makes a PWM sign depending on DIMMParam.
; If DIMMParam is 0, no PWM sign exist, the output pin is HIGH.

```

DoOut                          btfss SWStatus,PWMUpdate ; Does PWM need to update ?
                                retlw NULL              ; No
                                decfsz PWMCycleCount,F; Yes, Decrement PWM cycle

```

Electromechanical Switch Replacement

```

                                goto PWM Pulse          ;
                                bsf GPIO,PWMOut         ; Prepare ...
                                movf DIMMParam,W        ; ... next ...
                                movwf PWM PulseCount    ; ... pulse
                                bsf SWStatus,Pulse;
                                retlw NULL            ;
PWM Pulse          btfss SWStatus,Pulse      ; Need a pulse cycle ?
                                retlw NULL           ; No
                                decfsz PWM PulseCount,F ; Yes, do
                                retlw NULL           ;
                                bcf GPIO,PWMOut; clear output and ...
                                bcf SWStatus,Pulse    ; ... status
                                retlw NULL           ;
Delay              btfss SWStatus,RecInPr     ; Is receiving in progress ?
                                goto HalfCycle        ; No, polling rate is about 416 us
                                call HalfCycle;Yes, poll. rate is 833 us first half
                                movlw 0x10 ;
                                addwf TMR0,F;
                                call HalfCycle      ; Second half
                                bsf SWStatus,PWMUpdate;
                                retlw NULL           ;
HalfCycle         movf TMR0,W                ;
                                btfss STATUS,Z       ; Need to wait ?
                                goto HalfCycle        ; Yes
                                movlw HALFTIME       ; No
                                movwf TMR0 ; Reload
                                btfss SWStatus,PWMUpdate;
                                goto SetPWMUpdate;
ClearPWMUpdate    bcf SWStatus,PWMUpdate      ;
                                retlw NULL           ;
SetPWMUpdate      bsf SWStatus,PWMUpdate      ;
                                retlw NULL           ;
                                end                ;
-
```

Electromechanical Switch Replacement

NOTES:

Electromechanical Switch Replacement

NOTES:



Electromechanical Switch Replacement

Optical Pyrometer

*Author: Spehro Pefhany
Barclay Instruments Inc.
Burlington, Ontario, Canada
email: speff@interlog.com*

APPLICATION OPERATION

An optical pyrometer is a device that allows non-contact measurement of temperature in the range from less than 1000° F to more than 3000° F. It operates on the principle of comparing a hot filament against a background of the object to be measured using a simple optical system similar to half of a binocular.

Normally, these devices use no electronics at all, they consist of a power rheostat, an analog meter calibrated to read temperature and on/off switch. As portability is normally a requirement, a battery source such as alkaline 'C' or 'D' cells is used to power the device.

Incorporating a PIC12C5XX into an optical pyrometer has the following benefits:

- Improved battery life from using a PWM MOSFET to control power to the filament, thus saving typically half the energy consumed.
- Elimination of the power-wasting rheostat with two inexpensive momentary pushbuttons.
- Elimination of the on/off switch.
- Automatic power-off, thus saving the batteries should the user accidentally leave the filament on.

Only four external components are required to implement the above functions, including the momentary switches. The internal RC clock and reset circuitry of the PIC12C5XX are more than adequate for this application. The internal pull-ups on inputs and the wake-up on pin change functions are extremely useful in this application.

Functions

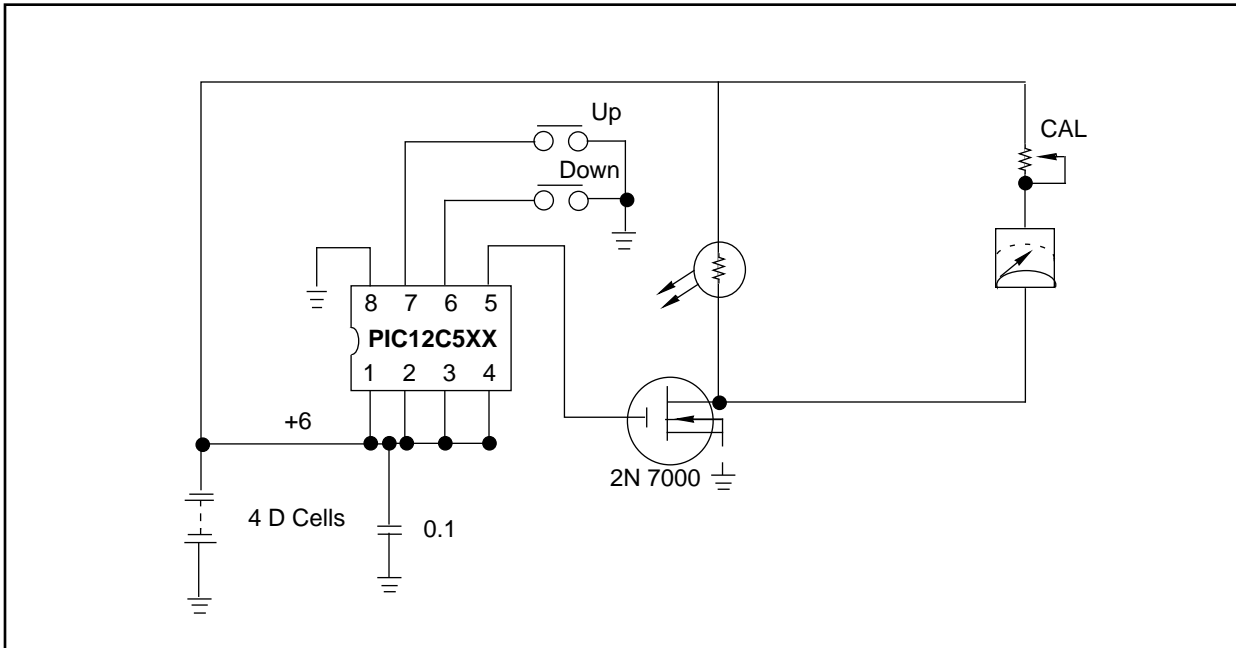
Pressing either the up or down key will "wake up" the PICmicro™ MCU and turn the unit on. The power to the filament is retained from the last measurement. The power can be adjusted up or down by holding down the appropriate key. Pressing both keys at once for more than a certain amount of time (0.8 second) will turn the unit off when the keys are released. If no keys are pressed for more than a certain amount of time (2.5 minutes), the unit will turn itself off. Power draw in the "off" condition is negligible compared to battery internal self-discharge leakage.

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.

Electromechanical Switch Replacement

GRAPHICAL HARDWARE PRESENTATION

The hardware used is shown below. No supply regulation is used since the PICmicro will operate over a wide range of supply voltage and the analog meter reads the actual average voltage on the filament. The weak pullup on GP3 also serves GP4 and GP5.



MICROCHIP TOOLS USED

Development Tools:

PISTART® Plus

Assembler/Compiler Version:

MPLAB 3.22, MPASM 1.5

Electromechanical Switch Replacement

APPENDIX A: SOURCE CODE

```
*****
;
; This program is for a PIC2C5XX microcontroller that will control a simple
; optical pyrometer using PWM and providing timed auto-off and on/off
; functions.
;
; (C) 1997 Spehro Pefhany, all rights reserved
;
; Rev. A
;
; - verified current in "off" state is less than 1uA
; - 15.544 msec execution for bigloop in simulator
;
*****
listp=12c509, r=HEX
include <P12C509.INC>; sfr definitions

#definePOWER_TIME0x26; desired time in seconds, divided by 3.96 (2.5min)
#define BOTHKEY_TIME0x1F; desired time in seconds, divided by 0.01554 (0.5 sec)

ctr equ 09 ; counter generates time ramp for comparison
c_valequ0A ; current value of PWM from 0..FF
repeatsequ0B; number of times to repeat the PWM loop
pwr_preequ0C; power off prescaler
pwr_ctrequ0D; power off counter
bothky_ctrequ0E; timer for both keys pressed to power off

    org 0

coldstart:
;*****
; Setup options
;*****
movwfOSCCAL; may as well use calibrated time
movlwH'00'

btfss7,STATUS; if a wake up , don't reset the set value
movwfc_val

movwfSTATUS; make sure that the page bit is cleared, of course it
; is supposed to be

movlwH'FB'; only GP2 is an output, other are inputs
trisGPIO; do it

movlwH'00'; set up for weak pullups and wake up on pin change
option
;*****
; Now set up the power off timer to full time
;*****
movlwH'FF'
movwfpwr_pre; set up prescaler
movlwPOWER_TIME
movwfpwr_ctr; set up counter
;*****
; And the bothky timer
;*****
movlwBOTHKEY_TIME
movwfbothky_ctr; set up counter
;*****
; The central PWM loop for controlling the filament intensity
;
; We only check for zero outside the tight inner loops
```

Electromechanical Switch Replacement

```
;*****
bigloop:
    movlwD'20'; number of times to repeat
    movwf repeats
    movfc_val,w; get the current value
    btfsc STATUS,2; skip if not zero
    goto iszero
replloop:
;*****
; Inner PWM loop.. if c_val is 0, don't even start
;
; The below gives exactly the same time to completion, regardless of
; on and off times, with some overhead where the output is off-- except
; for 0 input, must simulate that.
;*****
    movfc_val,w; do it again for the loop
    movwfctr
    bsfGPIO,2; turn the output on
onloop:
    decfszctr,f
    goto onloop
    bcfGPIO,2; turn the output off

    comfc_val,w; get complement of current value
    movwfctr
    incfctr,f; add 1 to it
offloop:
    decfszctr,f
    goto offloop
    decfszrepeats,f
    goto replloop; repeat multiple times
    goto continue
;*****
; Here we handle the special case of a zero c_val
;
; We even the time out exactly, even though it isn't critical in this case
;*****
iszero:
replloop1:
    bcfGPIO,2; just to be sure

    movlwH'00'; counter
    movwfctr
offloop1:
    decfszctr,f
    goto offloop1
    nop          ; even out the delay to make it exactly the
                ; same as the controlled on/off time loops

    nop
    nop          ; even it out exactly
    decfszrepeats,f
    goto replloop1; repeat multiple times
    nop          ; even time out exactly
continue:
;*****
; The "extra" delay loop for limiting the maximum light intensity
; with new batteries
;*****
; code goes in here, if required
;*****
; Now poll keys, check for time-out of power-off timer and
; do any key operations required.
;*****
    btfssGPIO,0; skip if no up key
    goto upkeyp
    btfssGPIO,1; skip if no down key
```

Electromechanical Switch Replacement

```
gotodownkeyp
;*****
; Check to see if we are in timeout situation on bothkey timer
;*****
movfbothky_ctr,f
btfscSTATUS,2; if not timed out, then continue
gotosnooze
;*****
; Reset the timer
;*****
movlwBOTHKEY_TIME
movwfbothky_ctr; reset the timer for both keys pressed

;*****
; There are no keys pressed, so count the power-off timer down and
; delay so it is the same as the other paths
;
; We hit this roughly every 16ms.. power off delay should be 2-3 minutes,
; so we need a count of 9,400 to get 2.5 minutes, a divide by 255 will
; give roughly 4 second per secondary count.
;
;*****
decfszpwpr_pre; count down prescaler
gotoback
gotodeccount
back:nop
nop
nop
nop
gotobigloop
deccount:
nop ; keep time same as other cycles
decfszpwpr_ctr,f; count down main counter
gotobigloop
;*****
; Now we have the power-off timer timeout, so we put the micro to
; sleep. It will wake upon a pin change.
;*****
snooze:
bcfGPIO,2 ; just to be sure
movfGPIO,w; read all pins, as manual recommends
sleep ; goes to reset on wake-up
;*****
; upkeyp handles an up key
;*****
upkeyp:btfsGPIO,1; this keeps time same as downkeyp as well
gotobothkeyp
comfc_val,w; see if it was FF
btfscSTATUS,2; skip if it wasn't FF
decfc_val,f
incfc_val,f
gotocommon1

;*****
; downkeyp handles a down key
;*****
downkeyp:
movfc_val,w; see if it is zero
btfscSTATUS,2; skip if it wasn't zero
incfc_val,f
decfc_val,f
gotocommon1

;*****
; bothkeyp handles situation where both keys are pressed
;
```

Electromechanical Switch Replacement

```
;*****
bothkeyp:
  movfbothky_ctr,w; check if already zero
  btfssSTATUS,2
  decfbothky_ctr,f; if not already zero, decrement
  nop          ; keep same time as others
  gotocommon
;*****
; Common ending for upkeyp and downkeyp and bothkeyp
;*****
common1:nop; keep all the times the same
common:
;*****
; Set up the power off timer to full time again
;*****
  movlwH'FF'
  movwfpwr_pre; set up prescaler
  movlwPOWER_TIME
  movwfpwr_ctr; set up counter

  gotobigloop

end
```



Electromechanical Switch Replacement

Programmable Lights

*Author: Kirill Yelizarov V.
Moscow Power Engineering
Institute
Moscow, Russia
email: tihonov@srv-vmss.mpei.ac.ru*

The switch itself has five buttons. Test key will start a test program called Test or program #0. Decrease Key or Increase Key will change an internal delay for lights change. Program Key will skip the program flow to the next program. If the last program line reached the first is fetched. Reset key will restore default settings (an internal delay for lights change) and start program #1.

I used a prototype board and handwired the circuit together. There are five buttons (S1-S5), a capacitor (C1), five resistors (R1 - R5), six diodes (D1-D6) and five red LEDs (D7-D11) on the board.

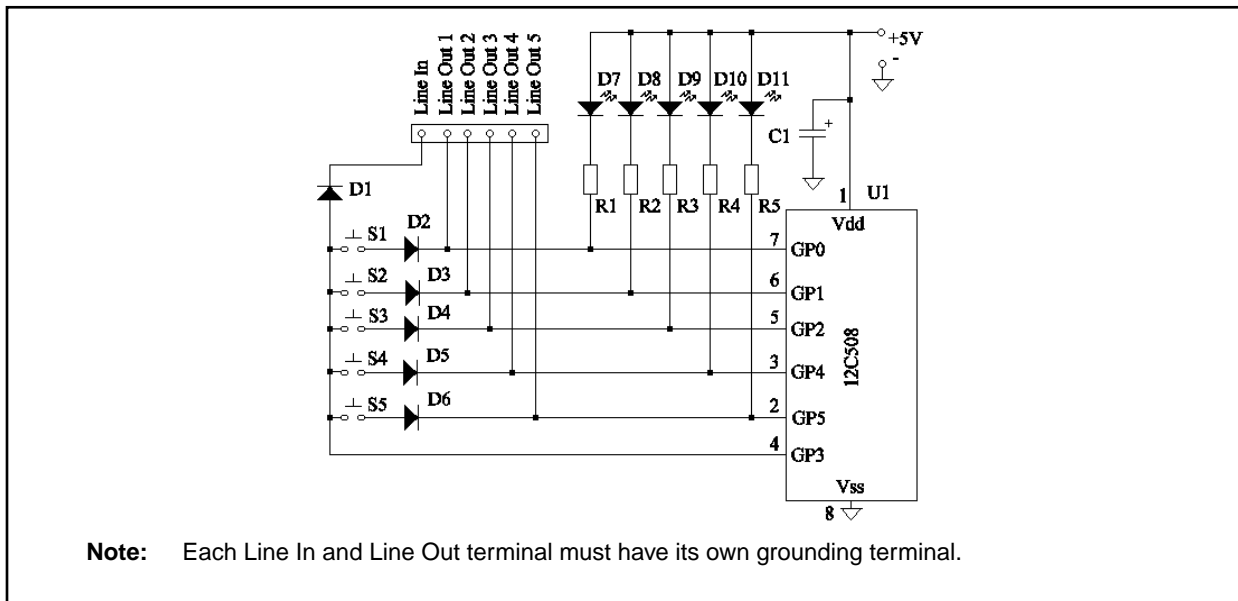
APPLICATION OPERATION

The electronic switch discussed in this application note may be used to operate five different appliances. It is based on the PIC12C508 and requires few external components. The program has a built in interpreter with loops which can output data to PICmicro pins. The language used in the interpreter has seven commands. The switch has five buttons to operate five outputs, one line-in terminal, and five line-out terminals.

With a few external components added, you can make Christmas lights for a Christmas tree. Attach five horns to your car, and they will play your favorite melodies. It is popular to animate car brake lights now. If connected to a timer via 'Line In', this switch can manipulate five appliances. Switches can be connected together through their "Line Out" and "Line In" terminals. This will increase the number of connected appliances.



FIGURE 1: SCHEMATIC DIAGRAM



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.

Electromechanical Switch Replacement

THE LIGHTS PARTS LIST

Capacitors:

C1 - 47 µf electrolytic

Diodes:

D1-D6 - Any type diodes

D7-D11 - Any type red light emitting diodes (LEDs may be replaced with Opto-relays or opto-triacs See Programmable Timer with Time Correction on how to connect to outlet. Special care should be taken when a transformerless power supply is used (see TB008) because any damage to grounding circuit will destroy ICs if two or more Programmable Lights are connected together.)

Resistors:

R1-R5 – depends on the type of LED

V_o = PIC12C508 output low voltage (0.6V Max)

V_{LED} = Input LED voltage (may vary from 0.8V IR to 2.0V green LED) (1.5 V red LED)

I_{LED} = LED current (10 mA Typical)

$$R1 = \frac{5 - V_{LED} - V_o}{I_{LED}} = 290\Omega$$

Miscellaneous

S1-S5 - normally open pushbutton switches

U1 - PIC12C508 programmed with Lights code

SOFTWARE

The program outputs the value of the Light command every 16 ms.

How the Keyboard Works.

First, all outputs (GP5, GP4, GP2, GP1, GP0) are raised high. Then, the first output GP0 (connected to button S1) is driven low (for 5 µs). On the third micro-second S1 is sampled. If the button is pressed, then GP3 is low. Each button has two key flags. Buttons are tested every 16 ms. If button is not pressed, then the flag is cleared. When the button is pressed and the flag is cleared, it is raised and the program continues to output lights. If the flag is raised already, then a required function is activated, and the second flag is raised else it is assumed to be a key debounce and flags are cleared. All buttons are tested sequentially. A low signal on the 'Line In' terminal will animate button S1 depression (this may be changed in code). To prevent reading buttons on polling of remote Programmable Lights connected to 'Line In', an additional check is made when all pins are high (this is needed if two appliances are absolutely synchronized).

PROGRAMMABLE LIGHTS MACRO LANGUAGE

Macro language used in this appliance has 7 commands. The are summarized in Table 1.

TABLE 1: TABLE OF COMMANDS

Command	Description
Light	Set new light
MarkLabel	Mark a label
ReturnToLabel	Return to label
SetRepeatValue	Set repeat value
JumpToProgram	Jump to program
StartOfProgram	Start of program
RestartProgram	Return to start of program

Light

This command is used to set microcontroller pins high or low. It is advisable to write this command like this:

```
Light b'00001'
```

Binary data send to PICmicro pins is (from right to left): GP5, GP4, GP2, GP1, GP0, where 0 sets the corresponding pin high, and 1 sets low. The command shown in the example will turn GP5, GP4, GP2 and GP1 high, and GP0 low.

SetRepeatValue MarkLabel RetunToLabel

These commands are used to make a loop with counter in the program. SetRepeateValue should precede MarkLabel command and is used to set the number of times the program will when ReturnToLabel command found roll over to MarkLabel. There is no default value, so each MarkLabel command must have a SetRepeatValue command. There may be 31 labels in the program. When a ReturnToLabel command is found the interpreter decreases repeat value and if it is not zero than it walks backward (this is significant, because in this case there may be more than 31 loops in the program) until a MarkLabel command is found. The repeat value should be in the range from 1 to 31 (If SetRepeatValue is set to 0 then commands inside loop will be output 256 times). An example shows how to write a loop correctly:

```
SetRepeatValue 10
MarkLabel 2
Light b'00001'
ReturnToLabel 2
```

This code example sets repeat value to 10 times and uses label #2. This code will set pin GP0 low ten times longer than the basic delay. Label numbers make no sense now but with some improvement to the code some label numbers may have unique counters and loops may be nested.

Electromechanical Switch Replacement

Start of Program RestartProgram

Part of program may be placed into an endless loop. To switch between endless loops this appliance has three buttons: Reset, Test and Program. There may be 31 loops. An example shows how it works:

```
StartOfProgram 5
.
.
.
StartOfProgram 1
Light b'00001'
RestartProgram 5
RestartProgram 1
```

If the current program number is #1 then command RestartProgram 5 is ignored. When RestartProgram 1 is fetched the interpreter walks forward until StartOfProgram 1 is found. Programs may be nested or crossed.

This commands may be useful to skip some code when several programs are nested or crossed. This is shown in the example:

```
StartOfProgram 1
Light b'00001'
RestartProgram 1
Light b'00100'
StartOfProgram 1
Light b'00010'
RestartProgram 1
```

In this case if the current program number is 1 then the interpreter will output 00001 and 00010 skipping 00100. This is because when RestartProgram is found the interpreter walks forward until the desired StartOfProgram is found.

Jump to Program

This command will jump to the desired StartOfProgram command. An example shows how it works:

```
.
.
.
StartOfProgram 1
Light b'00001'
RestartProgram 1
.
.
.
JumpToProgram 1
.
.
.
```

When JumpToProgram 1 is fetched the interpreter walks forward until StartOfProgram 1 is found. It is not the same as with RestartProgram since current program number may be not #1 when JumpToProgram 1 is fetched.

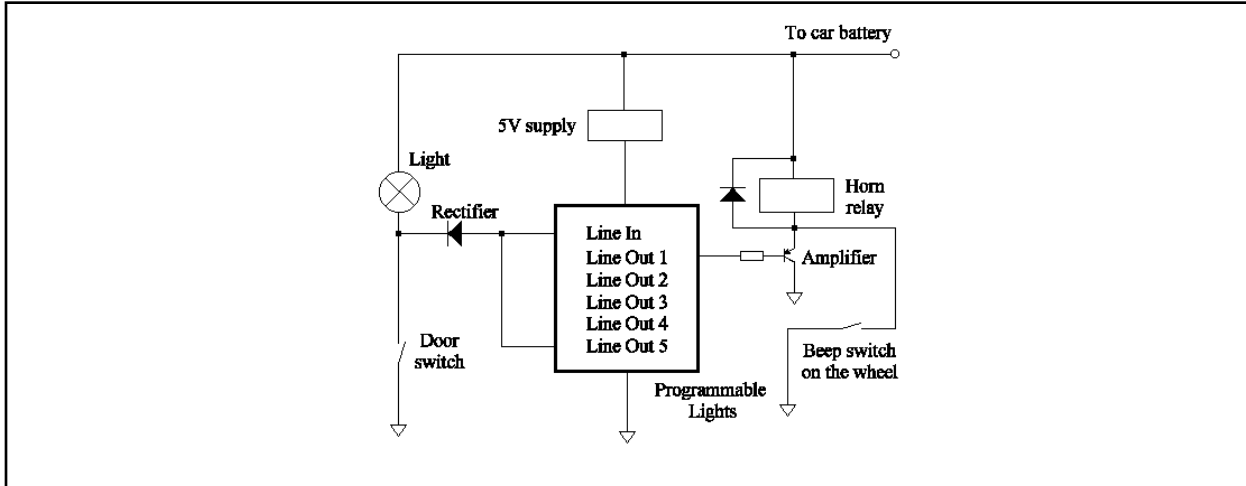
With the help of this command a simple alarm system bay be done. System block diagram is shown in figure 2. Look at the code:

```
define    MAX_PROGRAM=1
          StartOfProgram Test
          Lights b'00000'
          RestartProgram Test

          StartOfProgram 1
          SetRepeatValue 10
          MarkLabel 1
          Lights b'10001'
          Lights b'10000'
          ReturnToLabel 1
          JumpToProgram Test
```

Electromechanical Switch Replacement

FIGURE 2: CAR ALARM SYSTEM BLOCK DIAGRAM



Line in is connected to a car door open switch (normally closed contacts with one terminal connected to the ground) through a rectifier. GP0 is connected through an amplifier and a relay to a horn. GP5 is connected to line in. Other outputs maybe connected to parking lights and ignition immobilizer. Once the door is open this will change the current program number to #1 and the horn will beep 10 times. Line-in will be blocked with GP5 low signal to prevent several restarting of program number 1. Then the program will jump to test program and line in will be connected again. If the door is still opened there will be another beep. To switch the alarm off you need to push a Test button. If the thief tries to switch the car battery off and then on again the program will automatically start from program #1 and an alarm will occur.

QUICK CODE IDEAS

The code has a macro to test pressed buttons and a small function to read tables.

A macro to test pressed buttons is useful when the microcontroller has no interrupt flag on pin change and the only way to add a keyboard is to poll microcontrollers pins. This macro needs two bytes to store key flags (for 8 buttons or less).

I know several table read functions. Some are simple the rest are too long. This function is ideal for PIC12C508/509 since a call may be done only for the first 256 bytes of memory page code.

MICROCHIP TOOLS USED

Development Tools:

This code was written and debugged with MPLAB™ for Windows®/16 Version 3.22.02.

Assembler/Compiler version:

MPASM v01.50

Electromechanical Switch Replacement

APPENDIX A: SOURCE CODE

```
;Programmable Lights
;Author: Kirill Yelizarov

LIST P=PIC12C508, R=DEC
INCLUDE <p12c508.inc>

__CONFIG _IntRC_OSC & _WDT_OFF & _CP_OFF & _MCLRE_OFF

; ----- D A T A -----

MPC equ 0x07 ;Macro program counter
KeyFlags equ 0x08 ;Keyboard flags
Keys equ 0x09 ;Keyboard keys
Command equ 0x0a ;Current command
Value equ 0x0b ;Local data
Program equ 0x0c ;Current program number
TimeDelay equ 0x0d ;Time delay changed with Inc or Dec buttons
TimeCount equ 0x0e ;Current time count
RepeatCount equ 0x0f ;Counter for a loop

;----- KeyBoard bits -----

ResetKey equ 5 ;Switch to program #1
TestKey equ 4 ;Switch to program #0 (called Test)
LineIn equ 3 ;Line in for all buttons
DecKey equ 2 ;Decrease delay between lights flow
IncKey equ 1 ;Increase delay between lights flow
ProgKey equ 0 ;Switch to next program

; ----- Lights macro language -----
#define Light retlw 0x00+ ;Set new light
#define MarkLabel retlw 0x20+ ;Mark a label #(1...31)
#define ReturnToLabel retlw 0x40+ ;Return to label #(1...31)
#define SetRepeatValue retlw 0x80+ ;Set repeat value (1...31)
#define JumpToProgram retlw 0xa0+ ;Jump to program #(1...31)
#define StartOfProgram retlw 0xc0+ ;Start of program #(1...31)
#define RestartProgram retlw 0xe0+ ;Return to start of program #(1...31)

Mark equ 0x20
ReturnTo equ 0x40
RepeatValue equ 0x80
JumpTo equ 0xa0
StartOf equ 0xc0
Restart equ 0xe0

Button equ 0x00
Line equ 0x01

MAX_PROGMEQU equ 12
Test equ 0 ;Test program

; ----- M A C R O -----
;This macro is used to search pressed button
;total button de bounce test is 2*16 ms = 32 ms with one extra test on the 16th ms

TestButtonmacro source,line,butt,key,flag,out
local reset
local test

;source - type of signal (Button - synchronized by the lights, Line - not synchronized)
;line - input test GPIO pin
;butt - button output GPIO pin
;key - depression memory location
;flag - key flag memory location
;out - label to return from macro
```

Electromechanical Switch Replacement

```
        bcf      GPIO,butt      ;drive butt pin low
        nop
        btfsc   GPIO,line      ;look what happens on line pin
        goto    reset          ;if high go and reset all for this button
        bsf     GPIO,butt      ;drive butt pin high

        IF      source==Button ;extra test for synchronized buttons
        nop
        btfss   GPIO,line      ;test pin for high value
        goto    reset          ;if pin is still low then it's a line in signal
        ENDIF

        IF      source==Line
        nop
        btfsc   GPIO,line      ;test line in if signal is still low
        goto    reset          ;else a polling signal was read
        ENDIF

        btfsc   key,butt       ;test key depression for butt pin
        goto    test           ;yes there was a depression on last test, go to test
        bsf     key,butt       ;no then set key depression flag
        goto    out            ;and wait for the next test

reset:
        bsf     GPIO,butt      ;drive butt pin high
        bcf     key,butt       ;clear depression key
        bcf     flag,butt      ;clear key flag
        goto    out            ;test next

test:
        btfsc   flag,butt      ;test key flag
        goto    out            ;if set then test next (to prevent multiple run of the
                                ; same function while the button is pressed)
        bsf     flag,butt      ;set key flag
                                ;function code will be merged here

        endm

;          ----- C O D E -----

        org     0

        goto    Start          ;Skip subroutine and table

;This multifunction subroutine can fetch next, or previous, or current command from the table

DecPC          ;Get previous command
        decf    MPC,F
        movlw   low Table
        xorwf   MPC,W
        btfss   STATUS,Z
        goto    ReadPC
        movlw   low (Start-1)
        movwf   MPC
        goto    ReadPC

IncPC          ;Get next command
        incf    MPC,F
        movlw   low Start
        xorwf   MPC,W
        btfss   STATUS,Z
        goto    ReadPC
        movlw   low (Table+1)
        movwf   MPC

ReadPC        ;Read current command
        movf    MPC,W
        call    Table
        movwf   Value
```

Electromechanical Switch Replacement

```
retlw    0

;
;----- P R O G R A M -----
Table    ;Start of program table
movwf    PCL

;*****
;*      Christmas Lights      *
;*      1997                  *
;*      Author: Kirill Yelizarov *
;*****

StartOfProgram Test
Light    b'11111'
RestartProgram Test

StartOfProgram 1

Light    b'11111'
Light    b'11111'
Light    b'11111'
Light    b'01111'
Light    b'01111'
Light    b'01111'
Light    b'00111'
Light    b'00111'
Light    b'00111'
Light    b'00011'
Light    b'00011'
Light    b'00011'
Light    b'00001'
Light    b'00001'
Light    b'00001'

SetRepeatValue 10
MarkLabel 1
StartOfProgram 2
Light    b'10000'
Light    b'01000'
Light    b'00100'
Light    b'00010'
Light    b'00001'
RestartProgram 2
ReturnToLabel 1

SetRepeatValue 10
MarkLabel 2
StartOfProgram 3
Light    b'10001'
Light    b'11000'
Light    b'01100'
Light    b'00110'
Light    b'00011'
RestartProgram 3
ReturnToLabel 2

SetRepeatValue 10
MarkLabel 3
StartOfProgram 4
Light    b'10011'
Light    b'11001'
Light    b'11100'
Light    b'01110'
Light    b'00111'
RestartProgram 4
ReturnToLabel 3
```

Electromechanical Switch Replacement

```
SetRepeatValue 10
MarkLabel 4
StartOfProgram 5
Light      b'10111'
Light      b'11011'
Light      b'11101'
Light      b'11110'
Light      b'01111'
RestartProgram 5
ReturnToLabel 4
```

```
SetRepeatValue 10
MarkLabel 5
Light      b'00111'
Light      b'10011'
Light      b'11001'
Light      b'11100'
Light      b'01110'
ReturnToLabel 5
```

```
SetRepeatValue 10
MarkLabel 6
Light      b'00110'
Light      b'00011'
Light      b'10001'
Light      b'11000'
Light      b'01100'
ReturnToLabel 6
```

```
SetRepeatValue 10
MarkLabel 7
Light      b'00100'
Light      b'00010'
Light      b'00001'
Light      b'10000'
Light      b'01000'
ReturnToLabel 7
```

```
Light      b'00100'
Light      b'00010'
Light      b'00001'
```

```
StartOfProgram 6
Light      b'10000'
Light      b'10000'
Light      b'10000'
Light      b'11000'
Light      b'11000'
Light      b'11000'
Light      b'11100'
Light      b'11100'
Light      b'11100'
Light      b'11110'
Light      b'11110'
Light      b'11110'
Light      b'11111'
Light      b'11111'
Light      b'11111'
Light      b'11111'
Light      b'11110'
Light      b'11110'
Light      b'11110'
Light      b'11100'
Light      b'11100'
Light      b'11100'
Light      b'11000'
```

Electromechanical Switch Replacement

```
Light      b'11000'  
Light      b'11000'  
Light      b'10000'  
Light      b'10000'  
Light      b'10000'  
RestartProgram 6
```

```
SetRepeatValue 10  
MarkLabel 8  
StartOfProgram 7  
Light      b'00001'  
Light      b'00010'  
Light      b'00100'  
Light      b'01000'  
Light      b'10000'  
RestartProgram 7  
ReturnToLabel 8
```

```
SetRepeatValue 10  
MarkLabel 9  
StartOfProgram 8  
Light      b'10001'  
Light      b'00011'  
Light      b'00110'  
Light      b'01100'  
Light      b'11000'  
RestartProgram 8  
ReturnToLabel 9
```

```
SetRepeatValue 10  
MarkLabel 10  
StartOfProgram 9  
Light      b'11001'  
Light      b'10011'  
Light      b'00111'  
Light      b'01110'  
Light      b'11100'  
RestartProgram 9  
ReturnToLabel 10
```

```
SetRepeatValue 10  
MarkLabel 11  
StartOfProgram 10  
Light      b'11101'  
Light      b'11011'  
Light      b'10111'  
Light      b'01111'  
Light      b'11110'  
RestartProgram 10  
ReturnToLabel 11
```

```
SetRepeatValue 10  
MarkLabel 12  
Light      b'11100'  
Light      b'11001'  
Light      b'10011'  
Light      b'00111'  
Light      b'01110'  
ReturnToLabel 12
```

```
SetRepeatValue 10  
MarkLabel 13  
Light      b'01100'  
Light      b'11000'  
Light      b'10001'  
Light      b'00011'
```

Electromechanical Switch Replacement

```
Light      b'00110'  
ReturnToLabel 13  
  
SetRepeatValue 10  
MarkLabel 14  
Light      b'00100'  
Light      b'01000'  
Light      b'10000'  
Light      b'00001'  
Light      b'00010'  
ReturnToLabel 14  
  
Light      b'00100'  
Light      b'01000'  
Light      b'10000'
```

```
StartOfProgram 11  
Light      b'00001'  
Light      b'00001'  
Light      b'00001'  
Light      b'00011'  
Light      b'00011'  
Light      b'00011'  
Light      b'00111'  
Light      b'00111'  
Light      b'00111'  
Light      b'01111'  
Light      b'01111'  
Light      b'01111'  
RestartProgram 1  
Light      b'11111'  
Light      b'11111'  
Light      b'11111'  
Light      b'01111'  
Light      b'01111'  
Light      b'01111'  
Light      b'00111'  
Light      b'00111'  
Light      b'00111'  
Light      b'00011'  
Light      b'00011'  
Light      b'00011'  
Light      b'00011'  
Light      b'00001'  
Light      b'00001'  
Light      b'00001'  
RestartProgram 11
```

```
StartOfProgram 12  
Light      b'10001'  
Light      b'00000'  
Light      b'01010'  
Light      b'00000'  
Light      b'00100'  
Light      b'00000'  
Light      b'01010'  
Light      b'00000'  
RestartProgram 12
```

```
;  
Start:      ----- M A I N -----  
  
IF          Start>0x100  
ERROR      "Lights Message: Program Table too large."  
ENDIF  
  
clrf      Keys          ;Reset key depression
```

Electromechanical Switch Replacement

```
    clrf      KeyFlags      ;Reset keyboard flags
    clrf      TMR0          ;clear TMR0
    movlw    b'10000101'   ;Enable weak pull-up on GP3 and set prescaler to 1:64
;    movlw    b'10000000'   ;*****For MPLAB debug
    option

    clrf      GPIO
    comf     GPIO,F        ;Turn OFF the lights
    movlw    b'00001000'   ;Set GP3 (LineIn) as input and the rest are outputs
    tris     GPIO

    movlw    low (Start-1) ;Set program counter to the end of Table
    movwf    MPC

;    movlw    0x10
    movlw    0x01          ;*****For MPLAB debug
    movwf    TimeCount     ;Each time TMR0 overflows this value will decrement
    movwf    TimeDelay     ;Set time delay between two Light commands to 16*16 ms
                          ;= 256 ms may be changed with Inc and Dec keys

    movlw    0x01
    movwf    Program       ;Set program #1

SetProgram:
    call     IncPC         ;Get next command
    movf     Value,W
    andlw   b'11100000'
    xorlw   StartOf      ;If it's a StartOfProgram then continue
    btfss   STATUS,Z     ;else go and get another command
    goto    SetProgram
    movf     Value,W
    andlw   b'00011111'
    xorwf   Program,W    ;Be shure the right program found
    btfss   STATUS,Z     ;else go and get another command
    goto    SetProgram

MainLoop:
    call     IncPC         ;Get next command
    movf     Value,W
    andlw   b'11100000'   ;Test Light Command
    btfsc   STATUS,Z
    goto    SetLights
    movwf   Command      ;Save Command

    xorlw   ReturnTo     ;Test ReturnToLabel command
    btfss   STATUS,Z
    goto    TestRestartProgram ;else analyze next command
    movf     Value,W
    andlw   b'00011111'
    movwf   Command      ;Save label in Command because it's contents are no longer
needed

    decfsz  RepeatCount,F ;decrease loop counter set by
    goto    FindLabel
    goto    MainLoop

FindLabel:
    call     DecPC        ;Search backward for a desired label
    movf     Value,W
    andlw   b'11100000'
    xorlw   Mark
    btfss   STATUS,Z
    goto    FindLabel
    movlw   b'00011111'
    andwf   Value,W
    xorwf   Command,W
    btfss   STATUS,Z
    goto    FindLabel
    goto    MainLoop
```

Electromechanical Switch Replacement

```
TestRestartProgram: ;Look if it's a RestartProgram command
    movf      Command,W
    xorlw    Restart
    btfss    STATUS,Z
    goto     TestRepeatValue
    movlw    b'00011111'
    andwf    Value,W
    xorwf    Program,W
    btfss    STATUS,Z
    goto     MainLoop
    goto     SetProgram

TestRepeatValue:
    movf      Command,W
    xorlw    RepeatValue
    btfss    STATUS,Z
    goto     TestJumpTo
    movlw    b'00011111'
    andwf    Value,W
    movwf    RepeatCount
    goto     MainLoop

TestJumpTo:
    movf      Command,W
    xorlw    JumpTo
    btfss    STATUS,Z
    goto     MainLoop
    movlw    b'00011111'
    andwf    Value,W
    movwf    Program
    goto     SetProgram

SetLights:
    movlw    b'00110111'
    btfsc    Value,4
    bsf      Value,5
    bcf      Value,4
    btfsc    Value,3
    bsf      Value,4
    andwf    Value,F           ;cut everything but the lights

Wait1:
    btfss    TMR0,7           ;wait till seventh bit rise, comment it when in MPLAB debug
    goto     Wait1           ;***** Comment it when in MPLAB debug

    movlw    b'11111111'
    movwf    GPIO

    TestButton Button,LineIn,ProgKey,Keys,KeyFlags,TestInc

NextProgram:
    incf      Program,F
    movlw    MAX_PROGRAM+1
    xorwf    Program,W
    btfss    STATUS,Z
    goto     SetProgram
    movlw    0x01
    movwf    Program
    goto     SetProgram

TestInc:
    TestButton Button,LineIn,IncKey,Keys,KeyFlags,TestDec
    movlw    0x01
    movwf    TimeCount       ;Set TimeCount to one to make delay change fast
    incfsz   TimeDelay,F
    goto     TestDec
```

Electromechanical Switch Replacement

```
        movlw    0xff
        movwf    TimeDelay

TestDec:
        TestButton Button,LineIn,DecKey,Keys,KeyFlags,TestLine
        movlw    0x01
        movwf    TimeCount        ;Set TimeCount to one to make delay change fast
        decfsz   TimeDelay,F
        goto     TestLine
        movlw    0x01
        movwf    TimeDelay

TestLine:
        TestButton Line,LineIn,LineIn,Keys,KeyFlags,TestTest
        goto     NextProgram

TestTest:
        TestButton Button,LineIn,TestKey,Keys,KeyFlags,TestReset
        clrf     Program
        goto     SetProgram

TestReset:
        TestButton Button,LineIn,ResetKey,Keys,KeyFlags,RestoreLights
        goto     Start

RestoreLights:
        comf     Value,W
        movwf    GPIO        ;output Lights to GPIO

Wait0:
        btfsc   TMR0,7        ;wait till the seventh bit reset, *****Comment it when in
MPLAB debug
        goto     Wait0        ;*****Comment it when in MPLAB debug
        decfsz   TimeCount,F
        goto     Wait1        ;if zero not reached then make another key check
        movf     TimeDelay,W   ;reset TimeCount with TimeDelay value
        movwf    TimeCount
        goto     MainLoop

        org     0x1fff
        movlw    b'01110000'   ;set OSCCAL
        end
```



Electromechanical Switch Replacement

Replacing Electromechanical Switches

*Author: Lon Glazner
Solutions Cube
Chico, CA
email: solcubed@solutions-cubed.com*

DESIGN IDEA

The Smart Switch is the ultimate in electronic switch design. No switch has ever had such an immense impact on mankind's technical progress. It is assumed, and correctly so, that any device with such far reaching effects must be based on a product by Microchip Technology Inc. The Smart Switch provides a PWM output at 61Hz. This PWM output can be used to control pass elements or low-side switches (switches to ground).

In the case of pass elements this switch design could be used to produce variable DC voltages and/or high current sources through PNP pass elements. With LC filters and spike protection diodes the device could work well as a battery charger or even a step-down voltage regulator. In low cost applications the LC filters could be left off. It could also be useful as a volume control, or as an element in an automatic gain control circuit.

For extremely low cost designs, a switch to ground could be controlled with the PWM output. For easy interface, a logic level N-channel FET could easily be used to switch 80V at 8A to ground. This could allow the switch to control very large motors with varying speeds.

In this design, a low cost transistor supply is used to provide the current required to run the PIC12CXXX at 5V. Various transistors could be specified in this supply circuit based on the supply voltage accepted.



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.

Electromechanical Switch Replacement

HARDWARE METHODOLOGY

As discussed earlier, the basis for this design is a switching element that allows for adjustable control of various loads. Because the PIC12CXXX allows the designer to use the internal MCLR and the internal 4Mhz oscillator, this circuit can be built with a minimum of expense. Even the pull up resistors on the manual control switches could be replaced by the PIC12CXXX's internal pull ups. The generic aspects of the PWM circuit allow it to fit into many applications. In this particular example the control of the PWM duty cycle can occur in two ways. The first, and primary method, is through manual control. The Smart Switch polls the input switches. If the increase switch is pressed, it causes the PWM duty cycle to increment. If the decrease switch is pressed, it causes the PWM signal to decrement. The software debounce implemented is 15ms in duration. This debounce routine also serves as a typematic type function. In other words, if you continue to press one of the switches it will continuously increment or decrement the duty cycle. This can give the feel of smooth manual control of a load supply in designs where a potentiometer can't be used. The manual switches could easily be replaced with comparator outputs. In such a design, voltage thresholds could trigger a slewing of the duty cycle. This could be used to implement dual level float chargers for high capacity batteries or automated transition to a battery backup system.

The PWM output of 61Hz was chosen for software reasons, and because this frequency is about as slow as you can get and still generate smooth motor control with direct PWM.

The PIC12CXXX is powered by a zener diode controlled transistor supply. The emitter of the transistor will maintain a voltage of roughly 0.6V below the value of the zener diode. In this case it is roughly 5V.

Using the bill of materials below the Smart Switch could act as a 20A switch with a 35V supply. Package power dissipation for the logic level FET would have to be addressed.

TABLE 1: BILL OF MATERIALS

Designation	Part	Specifications
R1	51 ohm	5% 1/8 watt
R2	10K ohm	5% 1/8 watt
R3	10K ohm	5% 1/8 watt
R4	4.7K ohm	5% 1/8 watt
C1	10uF	10V
D1	5.6V	1/2 watt
Q1	2N3904	max 40V Vceo
Q4	F25N05L	50V @ 25A
U1	12C508-4/P	neato

SOFTWARE METHODOLOGY

The heart of the Smart Switch is the standard 15us delay. The usefulness of varying PWM outputs can be noted by the inclusion of PWM hardware in more complex PICmicro chips. If people didn't want it, it wouldn't be there. But many of Microchips customers still make use of the PIC16C5X cores for their designs. Multiplexing different time based functions without interrupts can be daunting, but with a little work, all PICmicros are capable of multitasking time based functions.

In this design, we started with an 15us standardized delay. Every routine in the program uses this delay. There is nothing sacred about the delay period. We set the TMR0 prescaler to 64us. The 15us delay simply checks TMR0 for a rollover. If a TMR0 rollover occurs the delay routine tests the output pin for the current logic level. It then loads the TMR0 register with either the positive duty cycle or the negative duty cycle duration and toggles the output pin. As long as the delay routine is called every 64us you can't miss a TMR0 rollover. And therefore the PWM output will remain constant.

With a standardized delay as a building block its easy to add other time based functions to the design. In the case of this design we added a 2400 baud serial interface that can be used to modify the duty cycle storage register. With this routine the PWM duty cycle could be continually updated based on the needs of your system. Additional I/Os are available to interface to EEPROM or to drive status LEDs. Microchip's new 16 byte EEPROM would be ideal for storage of the current duty cycle value, which would add the aspect of non-volatility to the Smart Switch.

RAM Used:..... 6 bytes
Subroutine Bytes:..... 110
Program Bytes (as presented): 139
Program Cycles (min, no PWM adjustments): 35
Program Cycles (max, switch
press/15ms debounce loop): 14,191

Electromechanical Switch Replacement

FIGURE 1: SMART SWITCH – ADJUSTABLE HIGH CURRENT PASS ELEMENT

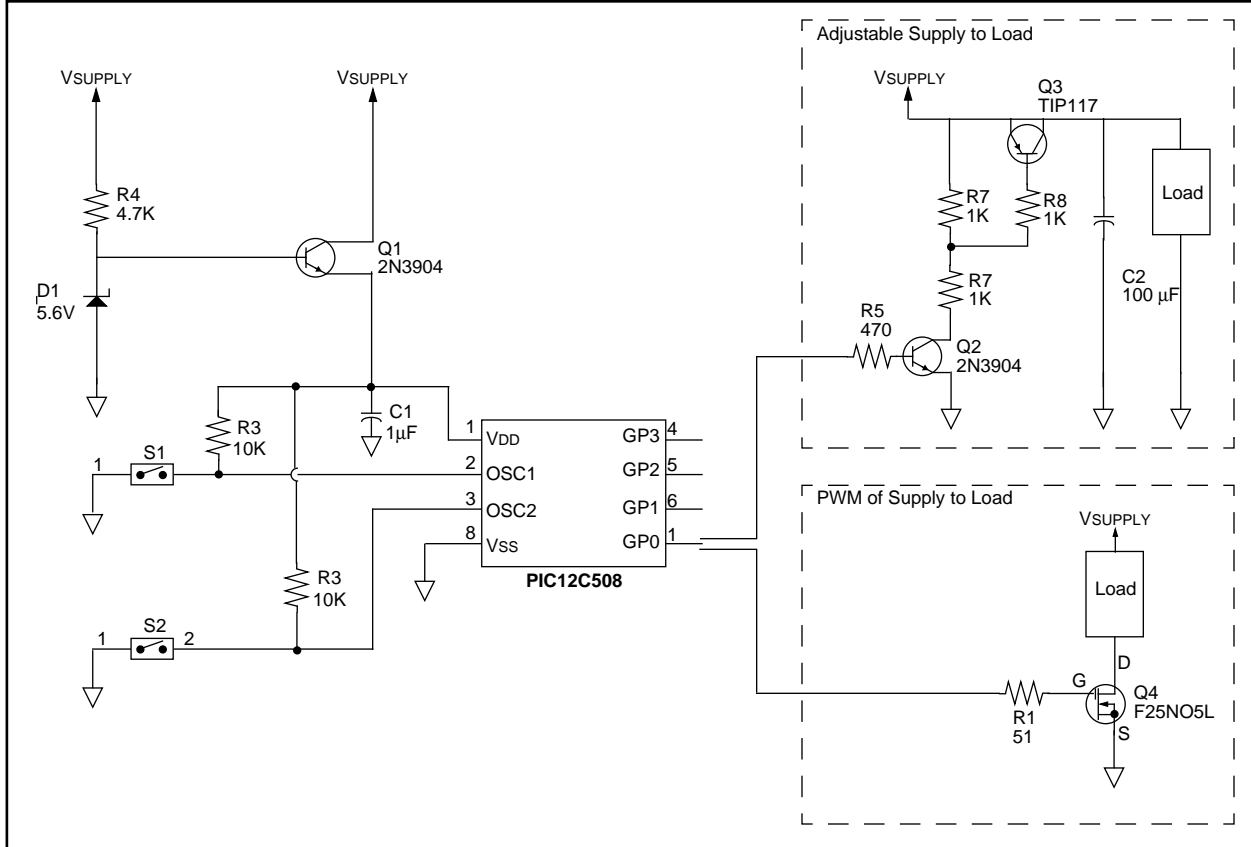
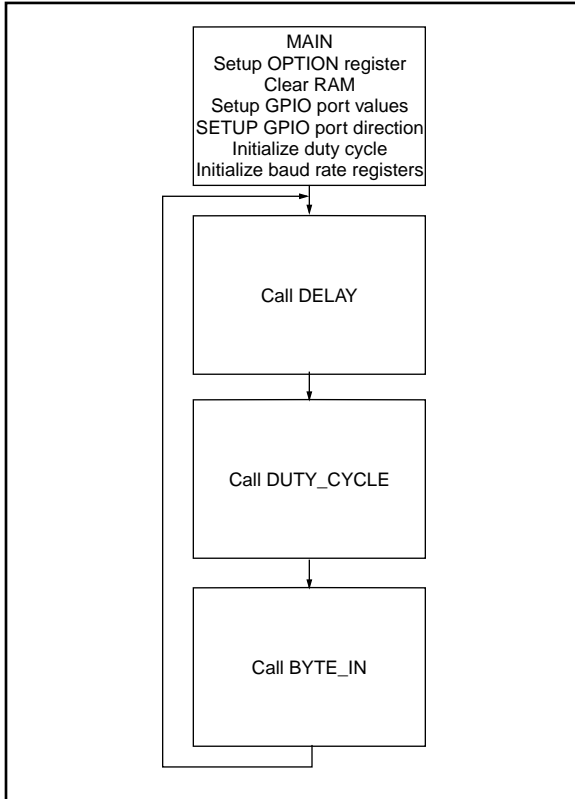
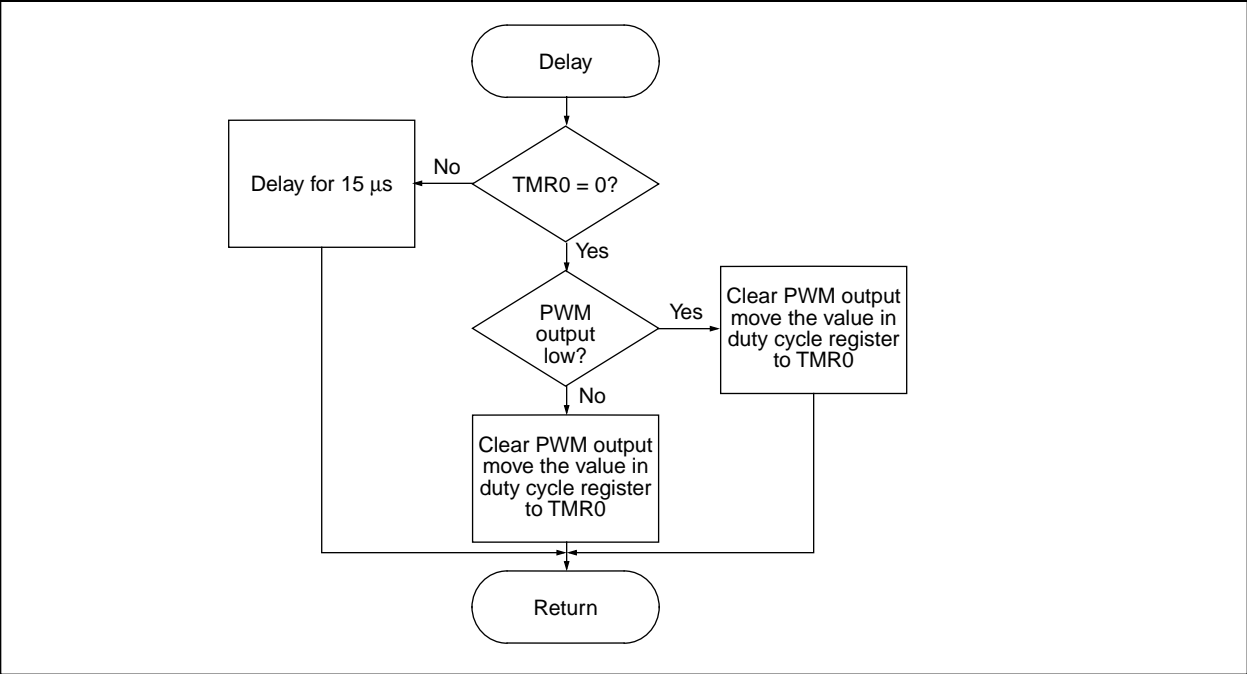


FIGURE 2: MAIN ROUTINE FLOWCHART



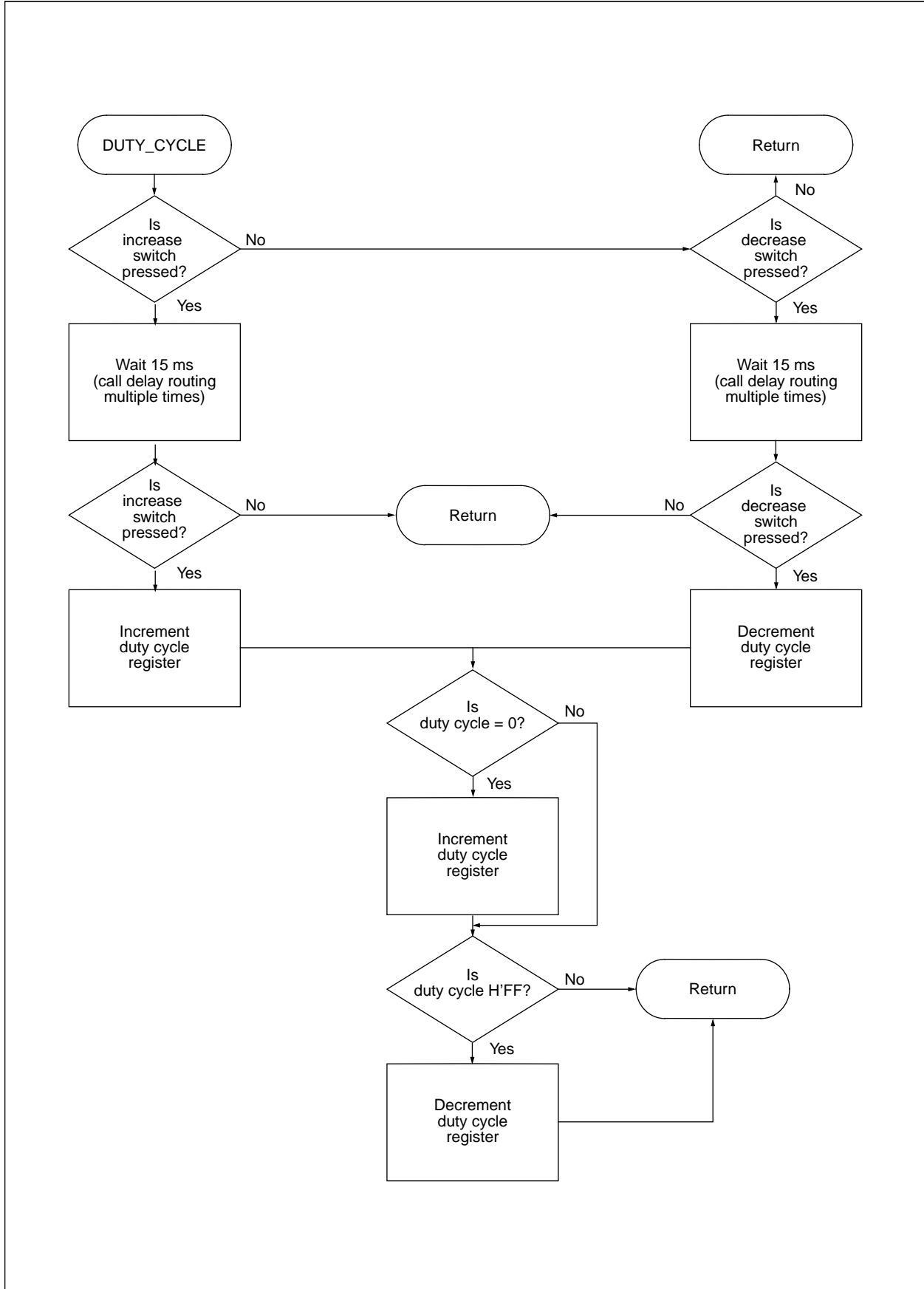
Electromechanical Switch Replacement

FIGURE 3: DELAY ROUTINE FLOWCHART



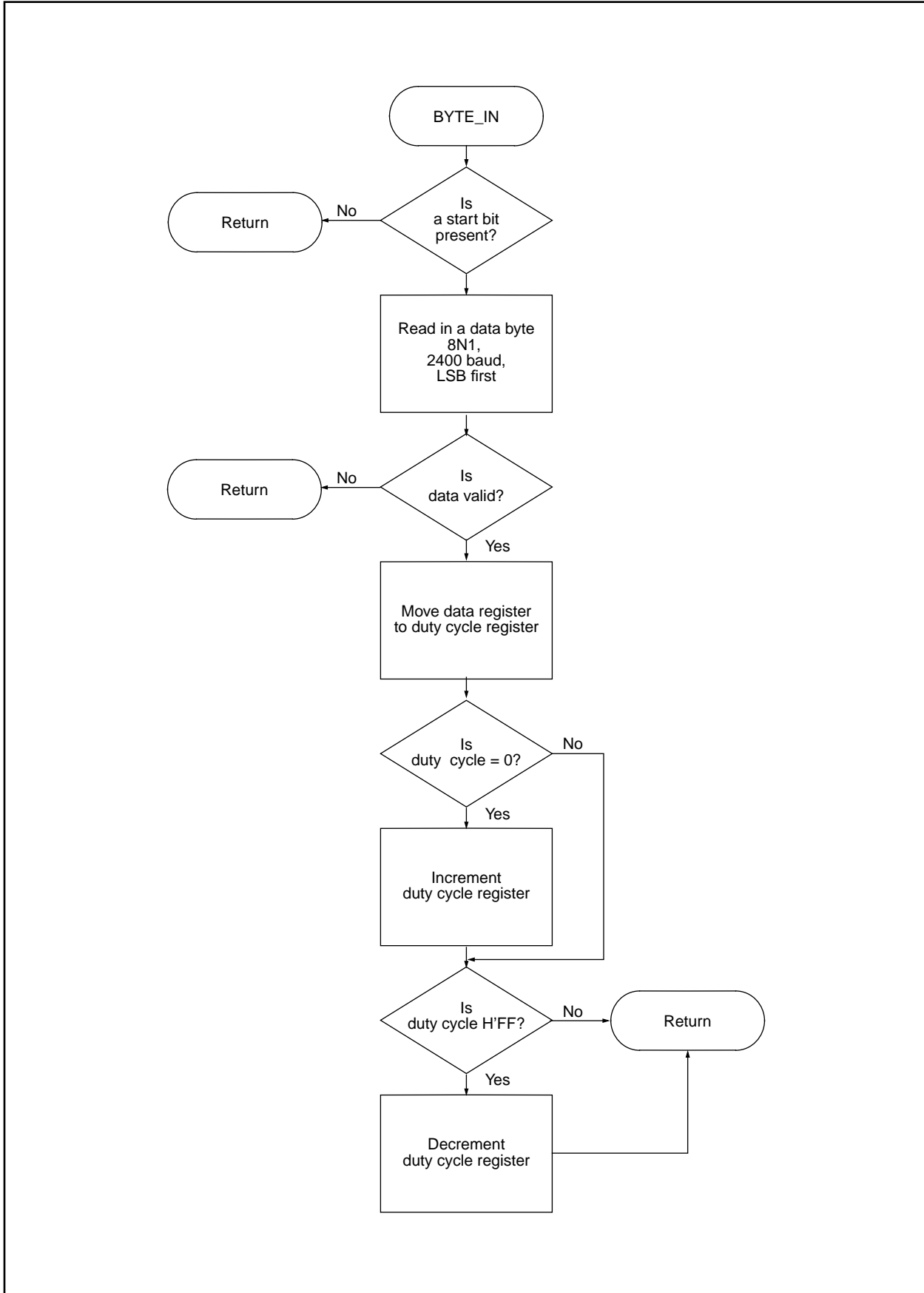
Electromechanical Switch Replacement

FIGURE 4: DUTY_CYCLE ROUTINE FLOWCHART



Electromechanical Switch Replacement

FIGURE 5: BYTE_IN ROUTINE FLOWCHART



Electromechanical Switch Replacement

APPENDIX A: SOURCE CODE

MPASM 01.40 Released

SMART3.ASM 6-13-1997 17:04:51

PAGE 1

```
LOC OBJECT CODE      LINE SOURCE TEXT
VALUE

00001 ;*****
00002 ;*****
00003 ;****      SOLUTIONS CUBED      ****
00004 ;****      Frank Rossini, Lon Glazner, David Brobst      ****
00005 ;*****
00006 ;*****
00007 ;
00008 ;
00009 ;*****
00010 ;**** Smart Switch Assembly Code Listing ****
00011 ;*****
00012 ;
00013 ;      The purpose of this code is to develop a pass element controller that
00014 ;      can be used to supply current to a load. Two hardware configurations
00015 ;      allow for shunting the load's low side to ground with a PWM signal,
00016 ;      or providing a variable DC voltage to a load with a pass element.
00017 ;      The current handling capabilities of the switch can be determined
00018 ;      by the chosen pass elements. The duty cycle adjustment switches may
00019 ;      be replaced by comparator input signals. The I/O's left over
00020 ;      could be used implement serial communication or data storage
00021 ;      in EEPROM. In this code example the constant 15us delay that provided
00022 ;      the backbone of the program is also used to implement serial
00023 ;      communication at 2400 baud.
00024 ;
00025 ;*****
00026 ;
00027 ;
00028 ;*****
00029 ;*****
00030 ;**** Define registers, constants, processor, and assembler directives ****
00031 ;*****
00032 ;*****
00033 ;
00034 ;Processor
00035 ;
00036      LIST      P=12C508      ;Processor used
00037 ;
00038      fuses:
00039 ;      WDT      - on
00040 ;      OSC      - internal RC
00041 ;      MCLR     - internal MCLR
00042 ;      CP      - code protect on
00043 ;
00044 ;Processor defined registers and bits
00045 ;
00046      INCLUDE "C:\PIC\HEADERS\P12C508.INC"      ;Microchip include file
00001      LIST
00002 ; P12C508.INC Standard Header File, Version 1.01      Microchip Technology, Inc.
00103      LIST
00048 ;Program defined registers
00049 ;
00000006 00050 GPIO      EQU      H'06'      ;Output port register
00000007 00051 TEMPO      EQU      H'07'      ;Temporary storage register
00000008 00052 TEMP1      EQU      H'08'      ;Temporary storage register
00000009 00053 DUTY_CYC      EQU      H'09'      ;Storage register for duty cycle
0000000A 00054 DATA_REG      EQU      H'0A'      ;Storage register for serial data
0000000B 00055 FULL_BIT      EQU      H'0B'      ;Holds full bit period delay
0000000C 00056 QURT_BIT      EQU      H'0C'      ;Holds a quarter bit period delay
```

Electromechanical Switch Replacement

```
00057
00058 ;
00059 ;Program defined bits
00060 ;
00061 ;GPIO port bits
00000000 00062 OUTPUT EQU H'00' ;GPIO, output enable pin
00000003 00063 INPUT EQU H'03' ;GPIO, input for serial data
00000004 00064 SW1 EQU H'04' ;GPIO, input switch #1
00000005 00065 SW2 EQU H'05' ;GPIO, input switch #2
00066 ;*****
00067 ;
00068 ;
00069 ;*****
00070 ;*****
00071 ;*** Reset Vector ***
00072 ;*****
00073 ;*****
0000 00074 ORG H'000'
0000 0025 00075 MOVWF OSCCAL ;Move internal trim value to osccal
0001 0A71 00076 GOTO MAIN
00077 ;*****
00078 ;*****
00079 ;BYTE_IN: Byte_in receives data from a master at 2400 baud, 8N1, LSB first.
00080 ; If a valid byte of data is read in then it replaces the current
00081 ; duty cycle value in the SMART SWITCH. In this way serial
00082 ; communication can control the duty cycle of the switch.
00083 ;
00084 ; Called From: MAIN
00085 ; Modified Registers: STATUS, TEMP0, TEMP1, GPIO, DATA_REG
00086 ; DUTY_CYC
00087 ; Subroutines Called: DELAY
00088 ; Enabled Interrupts: NONE
00089 ;
0002 00090 BYTE_IN
0002 0004 00091 CLRWDT
0003 0666 00092 BTFSC GPIO,INPUT ;Test for a start bit
0004 0800 00093 RETLW H'00'
0005 020C 00094 MOVF QURT_BIT,W ;Set up timer for start bit check
0006 0027 00095 MOVWF TEMP0
0007 00096 delay_loop
0007 095B 00097 CALL DELAY
0008 0666 00098 BTFSC GPIO,INPUT ;Make sure input remains low
0009 0800 00099 RETLW H'00'
000A 02E7 00100 DECFSZ TEMP0
000B 0A07 00101 goto delay_loop
000C 0C08 00102 MOVLW H'08' ;Set up temp to count 8 bits
000D 0028 00103 MOVWF TEMP1
000E 00104 more_bits
000E 020B 00105 MOVF FULL_BIT,W ;Load temp with baud rate value
000F 0027 00106 MOVWF TEMP0
0010 00107 first_bit
0010 095B 00108 CALL DELAY ;22*18us + 20us = 0.415ms ->2403 baud
0011 02E7 00109 DECFSZ TEMP0
0012 0A10 00110 goto first_bit
0013 0766 00111 BTFSS GPIO,INPUT ;Test input value
0014 0403 00112 BCF STATUS,C ;Input was low so clear carry bit
0015 0666 00113 BTFSC GPIO,INPUT
0016 0503 00114 BSF STATUS,C ;Input was high so set carry bit
0017 032A 00115 RRF DATA_REG ;Rotate carry bit into data reg.
0018 0000 00116 NOP
0019 0000 00117 NOP
001A 0000 00118 NOP
001B 0000 00119 NOP ;A little delay to get 2400 baud
001C 0000 00120 NOP
001D 0000 00121 NOP
001E 0000 00122 NOP
```

Electromechanical Switch Replacement

```
001F 0000    00123      NOP
0020 0000    00124      NOP
0021 0000    00125      NOP
0022 02E8    00126      DECFSZ  TEMP1
0023 0A0E    00127      goto    more_bits                ;1/(0.396ms + 20us) = .415 -> 2403baud
0024 020B    00128      MOVF    FULL_BIT,W
0025 0027    00129      MOVWF   TEMP0
0026 02A7    00130      INCF    TEMP0                    ;Add a little more to stop bit
0027                    00131  stop_bit
0027 095B    00132      CALL    DELAY                    ;23 * 18us = .414ms -> 2415 baud
0028 02E7    00133      DECFSZ  TEMP0
0029 0A27    00134      goto    stop_bit
002A 0766    00135      BTFSS   GPIO,INPUT              ;Test for valid stop bit
002B 0800    00136      RETLW   H'00'
002C 020A    00137      MOVF    DATA_REG,W             ;Move received data to W
002D 0029    00138      MOVWF   DUTY_CYC
002E 0643    00139      BTFSC   STATUS,Z                ;See if duty cycle is H'00'
002F 02A9    00140      INCF    DUTY_CYC                ;If so increment duty cycle
0030 0289    00141      INCF    DUTY_CYC,W             ;See if duty cycle is H'FF'
0031 0643    00142      BTFSC   STATUS,Z                ;See if duty cycle is H'FF'
0032 00E9    00143      DECF    DUTY_CYC                ;If so decrement duty cycle
0033 0800    00144      RETLW   H'00'
0034 0800    00145      RETLW   H'00'
00146 ;*****
00147 ;DUTY_CYCLE: This routine monitors the input pins for logic low levels and
00148 ; increments or decrements the current duty cycle depending on which
00149 ; switch is pressed. Afterwards the 15us delay routine will
00150 ; automatically adjust the duty cycle. The duty cycle is required to
00151 ; be from H'01' to H'FE'.
00152 ;
00153 ;      Called From:                MAIN
00154 ;      Modified Registers:         DUTY_CYC, STATUS
00155 ;      Subroutines Called:         DELAY
00156 ;      Enabled Interrupts:         NONE
00157 ;
0035                    00158  DUTY_CYCLE
0035 0686    00159      BTFSC   GPIO,SW1                ;Test for change of duty cycle
0036 0A45    00160      goto    check_decrement         ;Check next switch
0037 0C16    00161      MOVLW   H'16'                   ;Start switch debounce
0038 0028    00162      MOVWF   TEMP1
0039                    00163  debounce_loop_0
0039 0C20    00164      MOVLW   H'20'
003A 0027    00165      MOVWF   TEMP0
003B                    00166  debounce_loop_1
003B 0004    00167      CLRWDT
003C 095B    00168      CALL    DELAY                    ;Maintain current PWM output
003D 02E7    00169      DECFSZ  TEMP0                    ;Apply 15ms debounce to switch
003E 0A3B    00170      goto    debounce_loop_1
003F 02E8    00171      DECFSZ  TEMP1
0040 0A39    00172      goto    debounce_loop_0
0041 0686    00173      BTFSC   GPIO,SW1                ;Check switch after debounce
0042 0800    00174      RETLW   H'00'
0043 02A9    00175      INCF    DUTY_CYC                ;If good then increment duty cycle
0044 0A54    00176      goto    check_duty_thresholds
0045                    00177  check_decrement
0045 06A6    00178      BTFSC   GPIO,SW2                ;Test for change of duty cycle
0046 0800    00179      RETLW   H'00'
0047 0C16    00180      MOVLW   H'16'                   ;Start switch debounce
0048 0028    00181      MOVWF   TEMP1
0049                    00182  debounce_loop_2
0049 0C20    00183      MOVLW   H'20'
004A 0027    00184      MOVWF   TEMP0
004B                    00185  debounce_loop_3
004B 0004    00186      CLRWDT
004C 095B    00187      CALL    DELAY                    ;Maintain current PWM output
004D 02E7    00188      DECFSZ  TEMP0                    ;Apply 15ms debounce to switch
```

Electromechanical Switch Replacement

```
004E 0A4B    00189      goto      debounce_loop_3
004F 02E8    00190      DECFSZ   TEMP1
0050 0A49    00191      goto      debounce_loop_2
0051 06A6    00192      BTFSC   GPIO,SW2                ;Check switch after debounce
0052 0800    00193      RETLW   H'00'
0053 00E9    00194      DECF    DUTY_CYC                ;If good then decrement switch
0054          00195      check_duty_thresholds
0054 0209    00196      MOVF    DUTY_CYC,W
0055 0643    00197      BTFSC   STATUS,Z                ;See if duty cycle is H'00'
0056 02A9    00198      INCF    DUTY_CYC                ;If so increment duty cycle
0057 0289    00199      INCF    DUTY_CYC,W              ;See if duty cycle is H'FF'
0058 0643    00200      BTFSC   STATUS,Z
0059 00E9    00201      DECF    DUTY_CYC                ;If so decrement duty cycle
005A 0800    00202      RETLW   H'00'
00203 ;*****
00204 ;DELAY:  A standard 15us delay.  Instructions + loop + call and return
00205 ;        equal 15us.  If timer zero rolls over then the output
00206 ;        (GPIO,OUTPUT) is toggled.  The value in DUTY_CYC is the number
00207 ;        of 64us periods that GPIO,OUTPUT stays low.  The complement
00208 ;        of DUTY_CYC is the number of 64us periods that GPIO,OUTPUT stays
00209 ;        high.  This routine generally maintains the duty cycle of the
00210 ;        pass elements PWM.
00211 ;
00212 ;        Called From:           MAIN
00213 ;        Modified Registers:    TMR0, DUTY_CYC, GPIO, STATUS
00214 ;        Subroutines Called:    NONE
00215 ;        Enabled Interrupts:    NONE
00216 ;
005B 0201    00217      DELAY   MOVF    TMR0,W                ;Test for TMR0 rollover
005C 0743    00218      BTFSS   STATUS,Z
005D 0A69    00219      goto    hold                    ;If no rollover don't change output
005E 0606    00220      BTFSC   GPIO,OUTPUT
005F 0A65    00221      goto    t_clr
0060 0506    00222      BSF    GPIO,OUTPUT              ;Set output
0061 0249    00223      COMF    DUTY_CYC,W              ;Complement duty cycle to W
0062 0021    00224      MOVWF   TMR0                    ;Move value to TMR0
0063 0000    00225      NOP
0064 0A70    00226      goto    done                    ;Get out of routine
0065 0406    00227      t_clr   BCF    GPIO,OUTPUT        ;Clear output
0066 0209    00228      MOVF    DUTY_CYC,W              ;Move duty cycle reg to W
0067 0021    00229      MOVWF   TMR0                    ;Move value to TMR0
0068 0A70    00230      goto    done                    ;Get out of routine
0069 0000    00231      hold   NOP
006A 0000    00232      NOP
006B 0000    00233      NOP
006C 0000    00234      NOP
006D 0000    00235      NOP
006E 0000    00236      NOP
006F 0000    00237      NOP
0070 0800    00238      done   RETLW   H'00'
00239 ;*****
00240 ;****                               Main Program                               ****
00241 ;*****
0071          00242      MAIN
0071          00243      ;
0071          00244      OPTION_SETUP
0071 0CC5    00245      MOVLW   H'C5'                    ;1100 0101
0072 0002    00246      OPTION
0073          00247      CLEAR_REGISTERS
0073 0067    00248      CLRF    TEMPO                    ;Clear first RAM location for use
0074 0C18    00249      MOVLW   H'18'                    ;Number of registers to clear
0075 0027    00250      MOVWF   TEMPO
0076 0C08    00251      MOVLW   H'08'                    ;Start of RAM clearing
0077 0024    00252      MOVWF   FSR
0078          00253      clear_loop
0078 0060    00254      CLRF    INDF                    ;Clear register pointed to
```

Electromechanical Switch Replacement

```
0079 02A4    00255      INCF    FSR,F           ;Go to next RAM location to clear
007A 02E7    00256      DECFSZ  TEMP0,F        ;Check to see if all clearing done
007B 0A78    00257      goto   clear_loop
007C                    00258  PORT_SETUP
007C 0C3E    00259      MOVLW  H'3E'          ;0011 1110
007D 0026    00260      MOVWF  GPIO           ;Set output low
007E 0000    00261      NOP
007F 0C3E    00262      MOVLW  H'3E'          ;0011 1110
0080 0006    00263      TRIS   GPIO           ;Set GP0 direction as an output
0081                    00264  REGISTER_SETUP
0081 0C01    00265      MOVLW  H'01'          ;Start duty cycle at zero
0082 0029    00266      MOVWF  DUTY_CYC       ;(1 is as close to zero as possible)
0083 0C16    00267      MOVLW  H'16'          ;Start duty cycle at zero
0084 002B    00268      MOVWF  FULL_BIT       ;Initialize serial communication
0085 0C06    00269      MOVLW  H'06'          ;for 2400 baud
0086 002C    00270      MOVWF  QURT_BIT
0087 0061    00271      CLRF   TMRO
00272 ;*****
00273 ;*****
0088                    00274  MAIN_LOOP
0088 0004    00275      CLRWDT
0089 095B    00276      CALL   DELAY           ;Maintain PWM
008A 0935    00277      CALL   DUTY_CYCLE      ;Test for manual adjustment
008B 0902    00278      CALL   BYTE_IN         ;Test for serial data adjustment
008C 0A88    00279      GOTO   MAIN_LOOP       ;Do it all again
00280 ;*****
00281 ;
00282 ;End of code indicator
00283 ;
00284      END
```

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

```
0000 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX--- -----
```

All other memory blocks unused.

Program Memory Words Used: 141
Program Memory Words Free: 370

Errors : 0
Warnings : 0 reported, 0 suppressed
Messages : 16 reported, 0 suppressed



Electromechanical Switch Replacement

Smart Switch for Automotive Applications and More

*Author: Marc Hoffknecht
Aachen, Germany
email: hoffknecht@online.de*

INTELLIGENT PUSH BUTTON FOR AIR CONTROL AND MORE

When driving behind busses, big trucks or lorries, the ventilation system fills your car with stuffy exhaust fumes. Therefore, many cars are equipped with a mode to circulate the interior air. Usually, one forgets to disable this mode afterwards and the air is no longer replaced with fresh air. This smart switch solves the problem mentioned while maintaining full functionality of the existing system - utilizing a single push button.

(There is the same problem with the rear window defrosting system: The window usually is clean after a few minutes, but one always forgets to deactivate the defroster.)

APPLICATION OPERATION

Since a user usually needs the device (ventilation, defroster, etc.) only for a certain period of time, this application features a timer function.

STANDARD OPERATION

Press the push button to activate the device (a short beep and/or an LED going on will acknowledge this). After a defined period of time, the device is automatically deactivated (signaled by a short beep and five seconds of a blinking LED – showing the driver which device is currently off – in case there are both ventilation and defroster control or even more).

Note: So the 'interior air circulation' mode cannot be forgotten and will not stay active longer than necessary.

Alternatively, the driver can activate the device permanently by pressing the button for about one second (the PICmicro™ MCU will acknowledge this with a long beep and an LED going on). To switch the device off again, just hit the push button (signaled by a beep and an LED going off).

Note: The timer period may be interrupted by just pressing the button.

The actual usage is much more intuitive than this explanation might suggest (all one has to know is: short press - standard function, longer press - permanently on)! See flowchart.

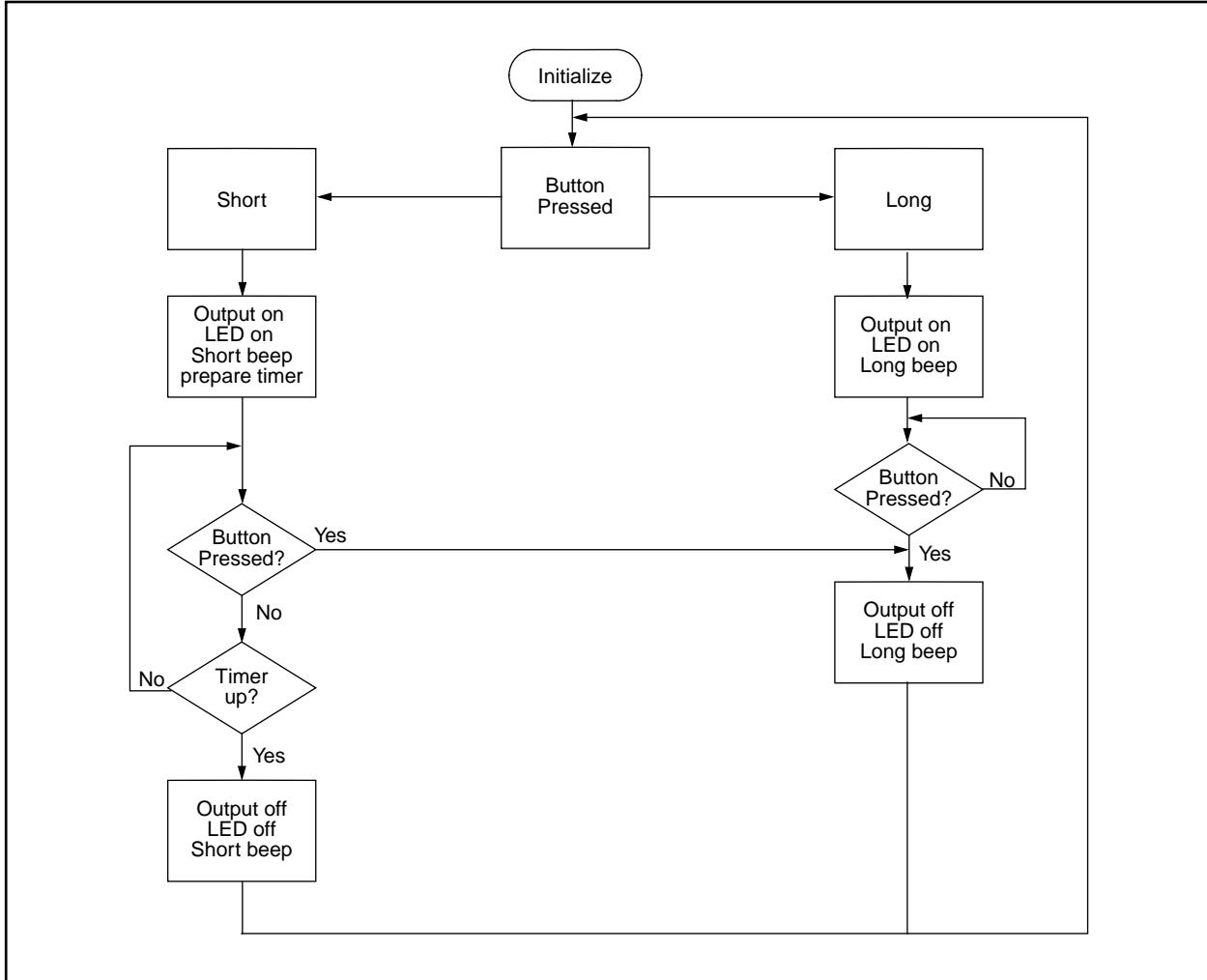
The software on the following pages show how to:

- Debounce buttons
- Implement an accurate timer with a period longer than is possible with the internal timer (very efficient code – a subroutine which just has to be called every now and then)
- Return boolean values (again very efficiently!)

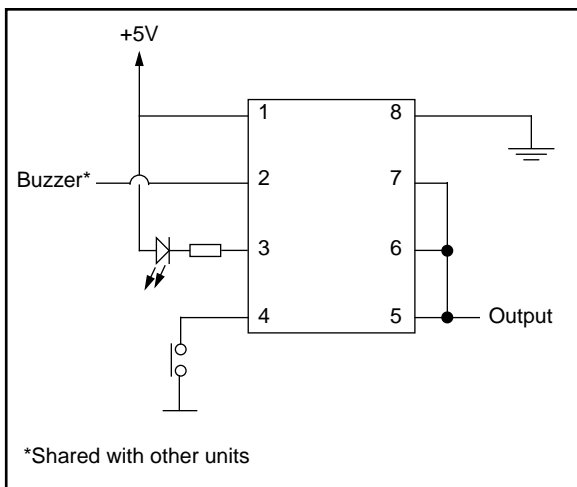
Microchip Technology Incorporated, has been granted a nonexclusive, worldwide license to reproduce, publish and distribute all submitted materials, in either original or edited form. The author has affirmed that this work is an original, unpublished work and that he/she owns all rights to such work.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

Electromechanical Switch Replacement

FLOW CHART



GRAPHICAL HARDWARE REPRESENTATION



Electromechanical Switch Replacement

APPENDIX A: SOURCE CODE

```
*****
;* Project: SmartSwitch
*****

processor 12c508
radix dec
include "p12c508.inc"

#define          __12C508
                __config _WDT_ON & _IntRC_OSC & _MCLRE_OFF & _CP_ON

#define zero     STATUS, 2
#define carry    STATUS, 0

#define TRUE     0
#define FALSE    -1

                CBLOCK 0x07      ; start of RAM
                ENDC

                GOTO Main

;* Hardware *****

#define Button   GPIO, 3
#define LED      GPIO, 5
#define Buzzer   GPIO, 4

LedOn           MACRO
                BCF LED; LED output is activ low
                ENDM

LedOff          MACRO
                BSF LED
                ENDM

BuzzerOn        MACRO
                BCF Buzzer; Buzzer is open drain in order
                MOVLW b'001000' ; to share it with other
                units
                TRIS GPIO
                ENDM

BuzzerOff       MACRO
                MOVLW b'011000'
                TRIS GPIO
                ENDM

OutputOn        MACRO
                MOVLW b'111000' ; output is activ low
                ANDWF GPIO
                ENDM

OutputOff       MACRO
                MOVLW b'000111'
                IORWF GPIO
                ENDM

#define          LongTime 800 ; Button down longer than this (in ms)
                ; will switch on permanently,
#define          Period 5*6000 ; otherwise output is this long on

;* Macros *****
TWSTrue         MACRO          ; (T)est (W) and (S)kip if (True)
                IORLW 0
                BTFSS zero
                ENDM

TWSTrue         MACRO
                IORLW 0
```

Electromechanical Switch Replacement

```

                                BTFSC zero
                                ENDM
TWSZero          MACRO
                                BTFSS zero
                                ENDM
TWSNonZero       MACRO
                                BTFSC zero
                                ENDM
RET              MACRO
                RETLW 0
                ENDM

;* Timer *****

; Usage: LoadTimer xx          ; Load timer with value xx ms
;           HandleTimer        ; execute this MACRO at least every
TMR0overrun us                ; and it will adjust the timer-variable
;                               ; and return true upon hitting zero
correctly
;
#define           TMR0overrun 16384          ; timer0 overrun every 16.4ms
                                                ; remember to change the option value
                                                ; in the main program when changing
                                                ; this

                                CBLOCK
                                TimerL
                                TimerH
                                OldTMR0
                                ENDC

HandleTimer_     MOVF OldTMR0, W            ; increase Timer on TMR0-overflow
                SUBWF TMR0, W             ; overflow, if OldTMR0 > TMR0
                BTFSC carry
                GOTO HT_done

                INCFSZ TimerL
                GOTO HT_done
                INCFSZ TimerH
                GOTO HT_done

                ADDWF OldTMR0
                RETLW TRUE                ; return TRUE upon hitting zero

HT_done         ADDWF OldTMR0
                RETLW FALSE

;

LoadTimer       MACRO Value
                MOVLW low(-1000*Value/TMR0overrun)
                MOVWF TimerL
                MOVLW high(-1000*Value/TMR0overrun)
                MOVWF TimerH
                ENDM

HandleTimer     MACRO
                ; The one routine is a MACRO, the
                CALL HandleTimer_; other a subroutine - who
can
                ENDM; remember ? So make both a MACRO !

;

Wait500ms       MACRO
                ; Due to the stack depth this is
                LOCAL Wait.loop; needed as a MACRO

Wait.loop      HandleTimer              LoadTimer 500

```

Electromechanical Switch Replacement

```

TWSTrue
GOTO Wait.loop
ENDM

;* Subroutines *****

                                CBLOCK
Counter                        ; for various counter-loops
                                ENDC

                                ;

ShortBeep      LoadTimer 200                ; 200ms beep
                                GOTO Beep

LongBeep       LoadTimer 800                ; 800ms beep

Beep           BTFSC TMR0, 2                ; this will generate about 2 kHz
                                BuzzerOn
                                BTFSS TMR0, 2
                                BuzzerOff
                                HandleTimer
                                TWSTrue
                                GOTO Beep
                                BuzzerOff
                                RET

                                ;

LedBlink       MOV LW 5
LedBlink.loop  LedOn

                                MOVWF Counter

                                Wait500ms
                                LedOff
                                Wait500ms
                                DECFSZ Counter
                                GOTO LedBlink.loop
                                RET

                                ;                                ; debounce buttons now

WaitTillPressed LoadTimer 50                ; reload timer with 50 ms ...
WTP.loop       BTFSC Button
                                ;
                                GOTO WaitTillPressed; ... while button not pressed
                                HandleTimer
                                TWSTrue
                                GOTO WTP.loop
                                RET

WaitTillReleased LoadTimer 50              ; reload timer with 50 ms ...
WTR.loop       BTFSS Button
                                ;
                                GOTO WaitTillReleased; ... while button pressed
                                HandleTimer
                                TWSTrue
                                GOTO WTR.loop
                                RET

;*****

Main           MOVWF OSCCAL

                                MOV LW b'10010101'; pullups on
                                OPTION                ; -> TMR0overrun every 16.384us

                                BuzzerOff            ; this will also set TRIS correctly
                                OutputOff
                                LedOff
```

Electromechanical Switch Replacement

```

;
Main.loop      CALL WaitTillPressed

               OutputOn
               LedOn

CheckMode      BTFSC Button                ; let's see whether button is held
               ;                          that long ...
               GOTO ModeShort
               HandleTimer
               TWSTrue
               GOTO CheckMode

;

ModeLong       CALL LongBeep                ; Mode: permanently on
               CALL WaitTillReleased; Button maybe still pressed
               CALL WaitTillPressed
               GOTO ManualOff

;

ModeShort      CALL ShortBeep                ; Mode: period on
               LoadTimer Period

ModeShort.loop BTFSS Button
               GOTO ManualOff
               HandleTimer
               TWSTrue
               GOTO ModeShort.loop

               OutputOff
               CALL ShortBeep
               CALL LedBlink
               GOTO Main.loop

;

ManualOff      OutputOff
               LedOff
               CALL LongBeep
               CALL WaitTillReleased
               GOTO Main.loop

;*****
END
```



Electromechanical Switch Replacement

Smart Switch for Car Windscreen Wiper Control

Author: Marc Hoffknecht
Aachen, Germany
email: hofknecht@online.de

PIC12C508 replaces potentiometer and multi-stage switch and increases user-friendliness.

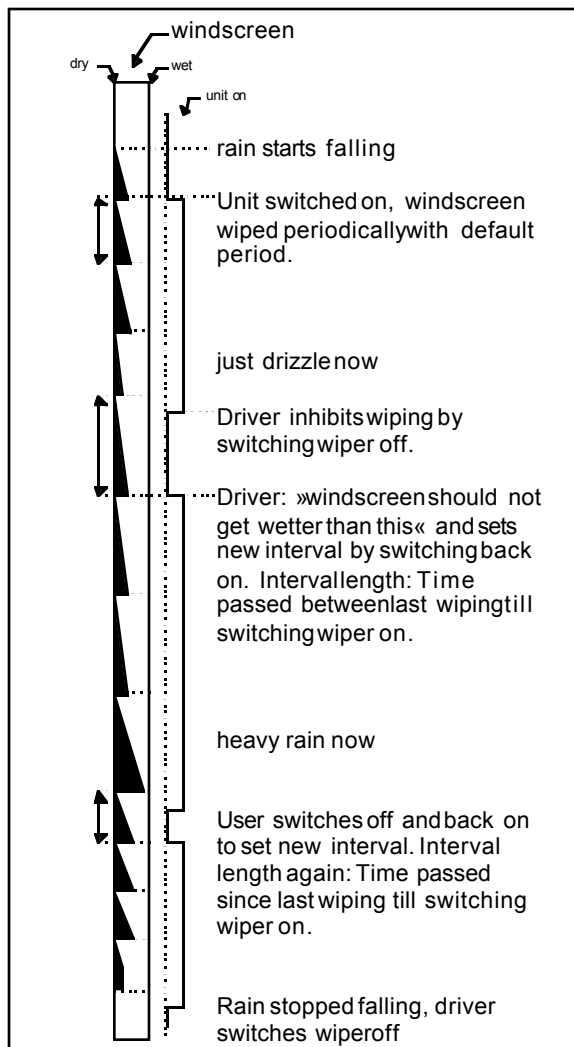
APPLICATION OPERATION

The usual wiper control in a car has two problems, both of which this application can solve. First, it uses too many parts, usually an on/off switch and either a potentiometer to adjust the wiping interval or a multi-stage switch. Second, it is not very user-friendly: You either have a limited number of interval periods or, if the wiper is controlled via a potentiometer, you have to adjust the interval period, watch the windscreen if the interval is sufficient (takes at least one or two times wiping), re-adjust the period and so on.

This application uses a single switch and a PIC12C508 to adjust the wiper interval settings. The main point is that the driver decides when the windscreen is too 'wet'. It is easiest to understand the operation using the attached graphics.

Upon switching the unit on, the windscreen is wiped periodically with a default interval. By switching the unit off, the driver inhibits wiping causing the windscreen to get wetter and wetter. As soon as the driver decides – the windscreen should not get wetter than this– he/she switches it back on. Doing so, the driver sets the new interval according to the time passed between the last clearing of the windscreen and switching the unit back on. This way the driver can either lengthen or shorten the interval to exactly what he wishes - the wipers will not go too fast (by the way, often it is the case in traffic jams, there's just no suitable setting!) and it won't go too slow.

OPERATION FLOWCHART



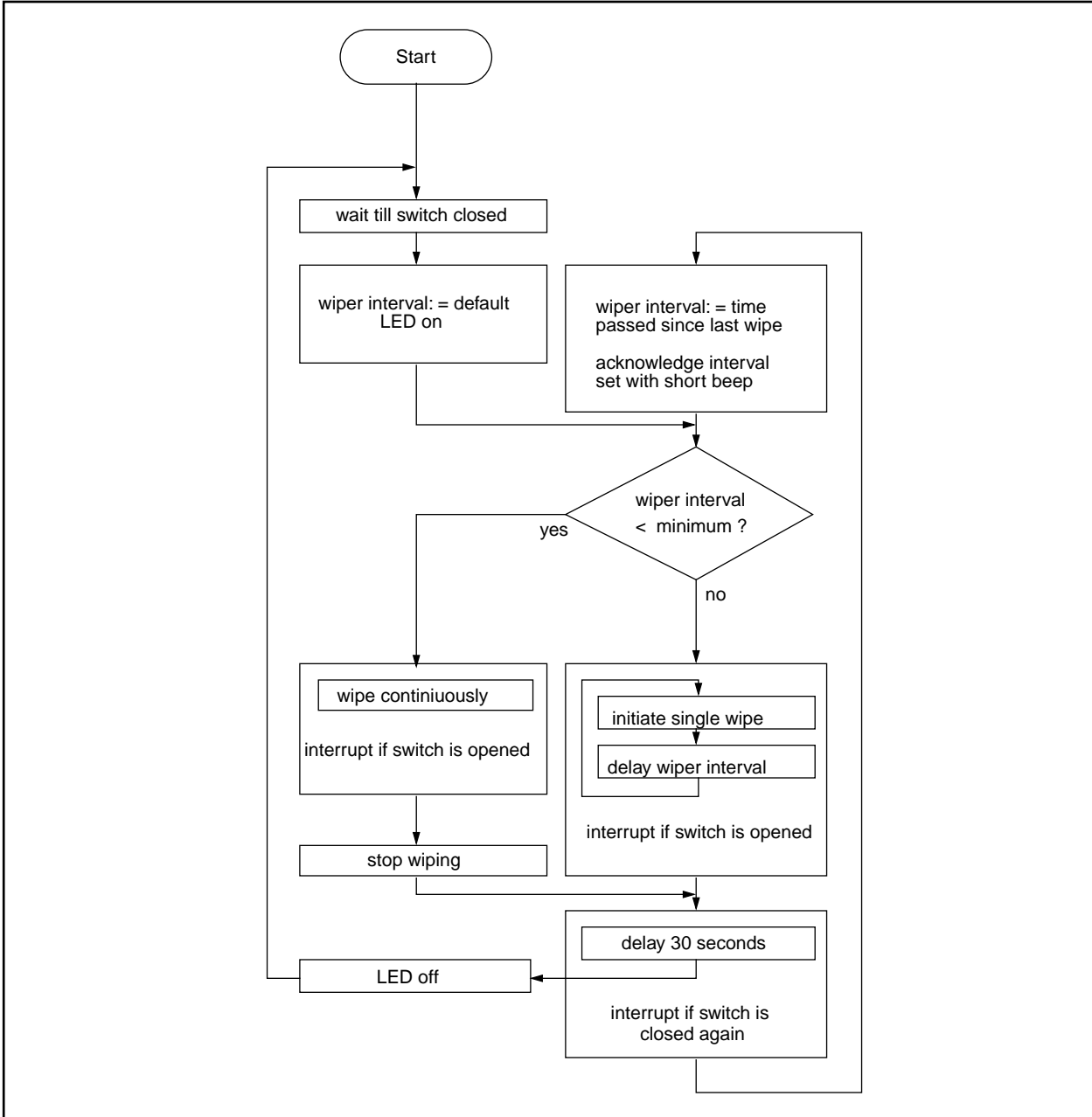
The software shows how to:

- Implement an accurate timer with a period longer than possible with the internal timer (very efficient code! a subroutine which just has to be called every now and then).
- Generate software interrupts.
- Debounce switches in an interrupt routine and protect the software against noise on the switch input.
- Return boolean values (again very efficiently!).

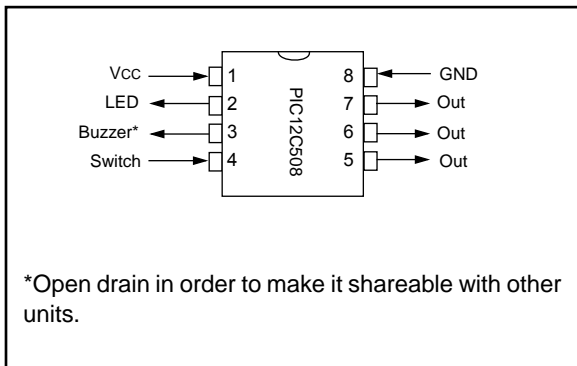
Microchip Technology Incorporated, has been granted a nonexclusive, worldwide license to reproduce, publish and distribute all submitted materials, in either original or edited form. The author has affirmed that this work is an original, unpublished work and that he/she owns all rights to such work.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

Electromechanical Switch Replacement

FLOW CHART



GRAPHICAL HARDWARE REPRESENTATION



MICROCHIP TOOLS USED

Assembler/Compiler Version:

MPASM V1.4

Electromechanical Switch Replacement

APPENDIX A: SOURCE CODE

```
*****
;* Project: Smart Wiper Switch                                     *
*****

        processor 12c508
        radix dec
        include "p12c508.inc"

#define          __12C508
        __config _WDT_OFF & _INTRC_OSC & _MCLR_OFF & _CP_ON

#define zero     STATUS, 2
#define carry    STATUS, 0

#define TRUE     0
#define FALSE    -1

        CBLOCK 0x07          ; start of RAM
        ENDC

        MOVWF OSCCAL
        GOTO Main

        ;

TMR0overrun    EQU 16384          ; timer0 overrun every 16.4ms
                                     ; remember to change the option value
#define ms      1000/TMR0overrun  ; in the main program when changing
#define secs    1000000/TMR0overrun ; this

;* Hardware *****

#define Switch      GPIO, 3
#define LED         GPIO, 5
#define Buzzer      GPIO, 4

#define LedOn      BCF LED          ; LED output is activ low
#define LedOff     BSF LED

BuzzerOn       MACRO
        BCF Buzzer          ; Buzzer is open drain in order
        MOVLW b'001000'     ; to share it with other units
        TRIS GPIO
        ENDM

BuzzerOff      MACRO
        MOVLW b'011000'
        TRIS GPIO
        ENDM

OutputOn       MACRO
        MOVLW b'111000'     ; output is activ low
        ANDWF GPIO
        ENDM

OutputOff      MACRO
        MOVLW b'000111'
        IORWF GPIO
        ENDM

        ;

WiperThreshold EQU 500*ms
MinimumInterval EQU 1*secs
DefaultInterval EQU 2*secs

BeepLength     EQU 200*ms
```

Electromechanical Switch Replacement

```
DenoiseTime          EQU 50*ms
DebounceTime        EQU 50*ms

DisableAfter         EQU 20*secs

;* Macros *****

TWSTrue              MACRO                ; (T)est (W) and (S)kip if (True)
                    IORLW 0
                    BTFSS zero
                    ENDM

TWSFalse            MACRO
                    IORLW 0
                    BTFSC zero
                    ENDM

#define              SkipIfZero BTFSS zero
#define              DoIfZero  BTFSC zero
#define              RET                RETLW 0

;* Switch *****

                    CBLOCK
                    Denoise
                    Debounce
                    Flags
                    ENDC
#define              SwitchClosed      Flags, 0

                    ;

HandleSwitch         MACRO
                    BTFSS Switch
                    GOTO HS.closed

HS.opened           MOVLW DebounceTime; switch open now, so
                    MOVWF Debounce    ; reset timer for 'switch closed'
                    BTFSS SwitchClosed
                    GOTO HS.done       ; switch is already denoised
                    DECFSZ Denoise     ; otherwise, wait till switch
                    GOTO HS.done       ; is stable a certain time
                    BCF SwitchClosed
                    GOTO HS.done

HS.closed           MOVLW DenoiseTime; as above
                    MOVWF Denoise
                    BTFSC SwitchClosed
                    GOTO HS.done
                    DECFSZ Debounce
                    GOTO HS.done
                    BSF SwitchClosed

HS.done             ENDM

;* Timer *****

                    CBLOCK
                    Timer0L
                    Timer0H
                    Timer1L
                    Timer1H
                    OldTMR0
                    ENDC

IncreaseTimer1      MACRO
                    INCFSZ Timer1L
                    GOTO Timer1.done
```

Electromechanical Switch Replacement

```

Timer1.done          INCF Timer1H
                    ENDM

Interrupt            MOVF OldTMR0, W          ; increase Timer on TMR0-overflow
                    SUBWF TMR0, W          ; overflow, if OldTMR0 > TMR0
                    BTFSC carry
                    GOTO Interrupt.done
                    ADDWF OldTMR0
                    ; program enters here every 16.4ms

                    HandleSwitch
                    IncreaseTimer1

                    INCFSZ Timer0L
                    RETLW FALSE
                    INCFSZ Timer0H
                    RETLW FALSE
                    RETLW TRUE          ; return TRUE upon hitting zero
                    ; in timer0 !

Interrupt.done      ADDWF OldTMR0
                    RETLW FALSE

                    ;

LoadTimer0          MACRO Value
                    MOVLW low(-Value)
                    MOVWF Timer0L
                    MOVLW high(-Value)
                    MOVWF Timer0H
                    ENDM

;* Subroutines *****

Beep                LoadTimer0 BeepLength
Beep.loop          BTFSC TMR0, 2          ; this will generate about 2 kHz
                    BuzzerOn
                    BTFSS TMR0, 2
                    BuzzerOff
                    CALL Interrupt
                    TWSTrue
                    GOTO Beep.loop
                    BuzzerOff
                    RET

                    ;

Delay              MACRO Value
                    LOCAL Loop
                    LoadTimer0 Value
Loop              CALL Interrupt
                    TWSTrue
                    GOTO Loop
                    ENDM

;*****

                    CBLOCK
                    Intervals
                    IntervalH
                    ENDC

Main              MOVLW b'10010101'; pullups on
                    OPTION          ; -> TMR0overrun every 16.384us

                    BuzzerOff          ; this will also set TRIS correctly
```

Electromechanical Switch Replacement

```

                                OutputOff

                                MOVW DebounceTime
                                MOVWF Debounce
                                BCF SwitchClosed

                                ;

Main.loop      LedOff

                                CALL Interrupt
                                BTFSS SwitchClosed
                                GOTO Main.loop

                                MOVW low(DefaultInterval); Interval:= DefaultInterval
                                MOVWF IntervalL      ;
                                MOVW high(DefaultInterval);
                                MOVWF IntervalH      ;
                                LedOn

CheckInterval  MOVW high(MinimumInterval); if Interval<MinimumInterval
                                SUBWF IntervalH, W ; then GOTO Wipe.continuous
                                BTFSS carry      ;
                                GOTO Wipe.continuous;
                                BTFSS zero      ;
                                GOTO Wipe.interval ;
                                MOVW low(MinimumInterval);
                                SUBWF IntervalL, W ;
                                BTFSS carry      ;
                                GOTO Wipe.continuous;

                                ;-----;

Wipe.interval  OutputOn          ; initiate single wipe
                                CLRF Timer1L      ; always clear timer1 upon
                                CLRF Timer1H      ; wiping
                                Delay WiperThreshold;
                                OutputOff        ;

Wipe.int.loop  CALL Interrupt

                                BTFSS SwitchClosed
                                GOTO Off?

                                MOVF IntervalH, W ; if timer1<Interval
                                SUBWF Timer1H, W ; then GOTO Wipe.int.loop
                                BTFSS carry      ; else GOTO Wipe.interval
                                GOTO Wipe.int.loop ;
                                BTFSS zero      ;
                                GOTO Wipe.interval ;
                                MOVF IntervalL, W ;
                                SUBWF Timer1L, W ;
                                BTFSS carry      ;
                                GOTO Wipe.int.loop ;
                                GOTO Wipe.interval ;

                                ;-----;

Wipe.continuous OutputOn

                                CLRF Timer1L      ; always clear timer1 upon
                                CLRF Timer1H      ; wiping
                                CALL Interrupt
                                BTFSS SwitchClosed
                                GOTO Wipe.continuous
                                OutputOff

                                ;-----;
```

Electromechanical Switch Replacement

```
Off?          LoadTimer0 DisableAfter
Off?.loop    BTFSC SwitchClosed
              GOTO NewIntervalSet
              CALL Interrupt
              TWSTrue
              GOTO Off?.loop
              GOTO Main.loop      ; 30 seconds expired

              ;-----;

NewIntervalSet  MOVF Timer1L, W
                MOVWF IntervalL
                MOVF Timer1H, W
                MOVWF IntervalH

                CALL Beep

                GOTO CheckInterval

;*****

                END
```



Electromechanical Switch Replacement

Smart Turn Signal Blinker

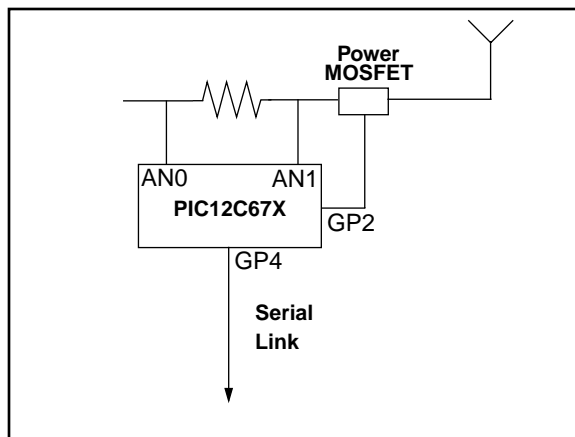
*Author: Kris Nielsen
Toshiba America IS / CSG
Irvine, CA
email: kris.nielsen@tais.toshiba.com*

OVERVIEW

Using the PIC12C67X, with a power MOSFET and some other circuitry, it could replace the typical thermal blinker. It could offer a large number of improvements. Using the ADC w/ current shunt, it could monitor for bulbs that had shorted or opened. Using one of the extra pins as a serial link, it could report back to a main computer in the car that one or more of the bulbs were bad. It would also offer short circuit protection by limiting current and shutting off the circuit. It would report the error back to the main computer in the car. It even could change the flashing frequency if necessary i.e. speed of car or weather/temp. With the addition of a small piezo speaker, it could offer a tone with it flash to remind people that the blinker was on. With additional inputs and mosfets, it could also monitor other lights on the vehicle, such as the brake lights. It could immediately warn the driver of bulb failures.

In order to correctly sense the current used by the light bulbs, it would need delay sensing the current a few 100ms after applying power to the bulbs. This is because of the surge current that the bulbs need. For the serial link, it would depend on the main computer in the car and what serial bus it uses. It could be a simple asynchronous port to a more complex I²C™ bus. The other functions are simple timing/delay loops.

BLOCK DIAGRAM



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.



MICROCHIP

WORLDWIDE SALES & SERVICE

AMERICAS

Corporate Office

Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602-786-7200 Fax: 602-786-7277
Technical Support: 602 786-7627
Web: <http://www.microchip.com>

Atlanta

Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston

Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

Chicago

Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas

Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 972-991-7177 Fax: 972-991-8588

Dayton

Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Los Angeles

Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 714-263-1888 Fax: 714-263-1338

New York

Microchip Technology Inc.
150 Motor Parkway, Suite 416
Hauppauge, NY 11788
Tel: 516-273-5305 Fax: 516-273-5335

San Jose

Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto

Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong

Microchip Asia Pacific
RM 3801B, Tower Two
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

India

Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-4036 Fax: 91-80-559-9840

Korea

Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai

Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hong Qiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700
Fax: 86 21-6275-5060

Singapore

Microchip Technology Taiwan
Singapore Branch
200 Middle Road
#07-02 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C

Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886 2-717-7175 Fax: 886-2-545-0139

EUROPE

United Kingdom

Arizona Microchip Technology Ltd.
Unit 6, The Courtyard
Meadow Bank, Furlong Road
Bourne End, Buckinghamshire SL8 5AJ
Tel: 44-1628-851077 Fax: 44-1628-850259

France

Arizona Microchip Technology SARL
Zone Industrielle de la Bonde
2 Rue du Buisson aux Fraises
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany

Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 München, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy

Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-39-6899939 Fax: 39-39-6899883

JAPAN

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa 222 Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

8/29/97

All rights reserved. © 1997, Microchip Technology Incorporated, USA. 9/97  Printed on recycled paper.

Information contained in this publication regarding device applications and the like is intended for suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights. The Microchip logo and name are registered trademarks of Microchip Technology Inc. in the U.S.A. and other countries. All rights reserved. All other trademarks mentioned herein are the property of their respective companies.



Electromechanical Switch Replacement

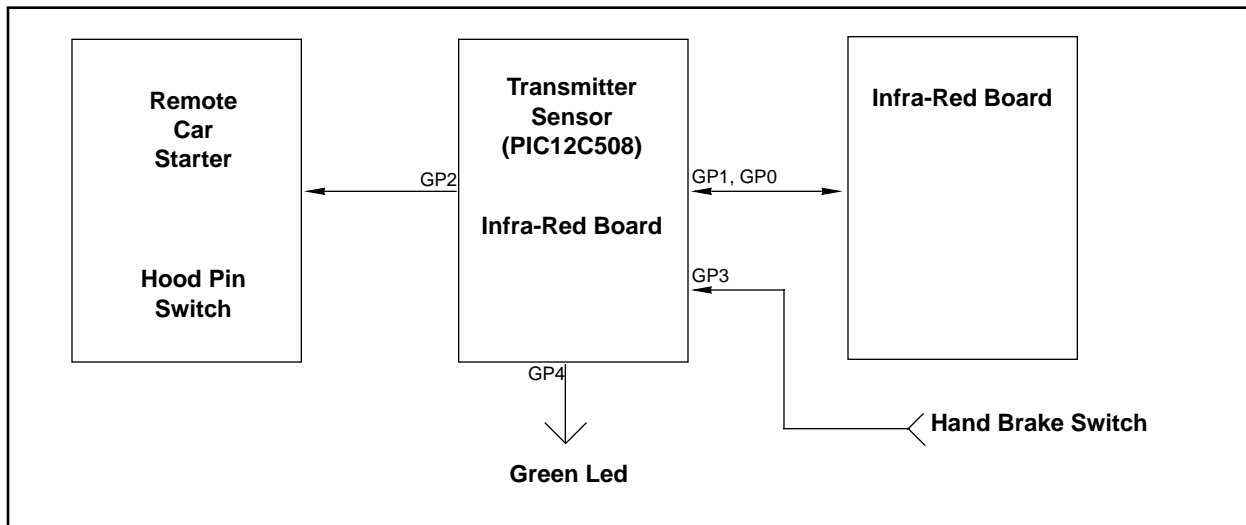
Transmission Sensor for Remote Car Starter

Author: Philippe Labonne
Shawinigan College
Grand-Mere, Quebec, Canada
email: destroy@infoteck.qc.ca

APPLICATION OPERATION

This application is a transmission sensor for a car with a remote starter. Because it is an automatic car remote starter, it is relatively cheap while one for a manual car can be very expensive. The principle behind the application is very simple. There are two infra-red LED emitting lights in-line with two phototransistors so that when the shifter is engaged, it must cut one of the two rays. There is an additional Hand Brake sensor for most cars having the same systems (a pin switch that touches the body when the hand brake is up). There is also a green LED connected to the microcontroller that, when all conditions are good (Shifter in middle, Hand Brake On), the LED blinks five times to show the driver that the car is ready for starting from the remote Starter.

BLOCK DIAGRAM



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.

Electromechanical Switch Replacement

GRAPHICAL HARDWARE REPRESENTATION

SENSOR.HPP = Schematic in HP Printer format

SENSOR.PCB = Board in PCAD format

SOLD1X.HPP = Solder side of PCB in HP Printer
Format (1X)

COMP1X.HPP = Components side of PCB in
HP Printer Format (1X)

SOLD2X.HPP = Solder side of PCB in HP Printer
Format (2X)

COMP2X.HPP = Components side of PCB in
HP Printer Format (2X)

MICROCHIP TOOLS USED

MPLAB version 1.4

Electromechanical Switch Replacement

APPENDIX A: SOURCE CODE

```
; Transmission Sensor For Remote Car Starter
; Philippe Labonne (PLAB97)
; Start : 5/6/97
; End   : 24/6/97

LIST      P=12C508
INDF      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'
COUNTER   EQU      H'0007' ; Counter for wait routine
ACC       EQU      H'0008' ; Accumulator
REG       EQU      H'0009' ; Var. for previos state

MOVWLW    02Ch                ; I/O Pin direction :
TRIS      GPIO                ; OOIIOI
CLRF      REG
CLRF      ACC
CLRF      COUNTER

Boucle    CALL      Delay1
          CALL      Delay1

          BSF      GPIO,1      ; Set the Infra-Red led ON
          CALL     Delay        ; Rise time
          BTFSS   GPIO,2      ; Look if CAP 1 is OK
          GOTO    Stop        ; Else Stop
          BTFSS   GPIO,3      ; Look if CAP 2 is OK
          GOTO    Stop        ; Else Stop
          BCF     GPIO,1      ; Set the Infra-Red led OFF

          BTFSC   GPIO,5      ; Look if Hand Brake is OK
          GOTO    Stop        ; Else Stop

          BTFSC   REG,0       ; Look if not allready Authorised to start
          GOTO    Boucle     ; If yes : start the loop again

          BSF     GPIO,0      ; Authorise the car to start
          BSF     REG,0       ; Save the state

          BSF     GPIO,4      ; Turn Green Led ON
          CALL    Delay1     ; Temp On
          BCF     GPIO,4      ; Turn Green Led OFF
          CALL    Delay1     ; Temp Off
          BSF     GPIO,4      ; Turn Green Led ON
          CALL    Delay1     ; Temp On
          BCF     GPIO,4      ; Turn Green Led OFF
          CALL    Delay1     ; Temp Off
          BSF     GPIO,4      ; Turn Green Led ON
          CALL    Delay1     ; Temp On
          BCF     GPIO,4      ; Turn Green Led OFF
          CALL    Delay1     ; Temp Off
          BSF     GPIO,4      ; Turn Green Led ON
          CALL    Delay1     ; Temp On
          BCF     GPIO,4      ; Turn Green Led OFF
          CALL    Delay1     ; Temp Off
          BSF     GPIO,4      ; Turn Green Led ON
          CALL    Delay1     ; Temp On
          BCF     GPIO,4      ; Turn Green Led OFF

          GOTO    Boucle
```

Electromechanical Switch Replacement

```
;----- Routine Stop si pas correct pour demmarer ----
Stop      BCF      GPIO,1      ; Turn infra-red led OFF
          BCF      GPIO,0      ; Stop Car
          BCF      REG,0       ; Save Car state
          BCF      REG,1       ; Save Green Led State
          CALL     Delay1      ; Wait
          GOTO     Boucle      ; Restart the loop
```

```
;----- Short Time Delay -----
Delay     MOVLW    040h
          MOVWF    ACC
Innrlp    MOVLW    0FFh
          MOVWF    COUNTER
Inlp      DECFSZ   COUNTER,1
          GOTO     Inlp
          DECFSZ   ACC,1
          GOTO     Innrlp
          RETLW    000h
```

```
;----- Long Time Delay -----
Delay1    MOVLW    0FFh
          MOVWF    ACC
Innrlp1   MOVLW    0FFh
          MOVWF    COUNTER
Inlp1     DECFSZ   COUNTER,1
          GOTO     Inlp1
          DECFSZ   ACC,1
          GOTO     Innrlp1
          RETLW    000h

          END
```

Electromechanical Switch Replacement

NOTES:

Electromechanical Switch Replacement

NOTES:

Electromechanical Switch Replacement

NOTES:



Electromechanical Switch Replacement

A Triple Input, Inverting Debounce Circuit

*Author: Jim Nagy
London, Ontario, Canada
email: nagy@wwdc.com*

APPLICATION OPERATION

This program configures the PIC12C508 as an interface between mechanical contacts and electronic circuits. This program creates a 'triple input, inverting debounce circuit', that requires no support circuitry other than the input contacts and a 2.5 to 5.5V supply.

Often several R-C networks and schmitt trigger gates are combined discretely to provide this function. But by using the internal RC oscillator, internal pullup resistors and software time delay circuits, I was able to create the interface entirely within the PIC12C508. One other advantage to using this chip is the ability to use the sleep mode to reduce power consumption when inputs are not changing.

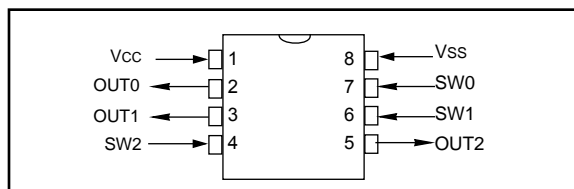
After the mechanical contacts have been processed by my circuit, the logic levels are clean and free of the bounce or jitter normally produced by a mechanical interface. High speed electronic circuits can then process the signals, without fear that the input is just random noise, or that bounces may cause multiple triggers.

The program was developed and assembled on a Macintosh computer, so I have followed IEEE syntax rather than MPASM.

GRAPHICAL HARDWARE REPRESENTATION

The PIC12C508 is connected as follows. Input contacts are connected between circuit common (Vss) and the inputs. Although the sample program only inverts the inputs, any combination could actually be used to control the outputs, which control electrical circuits/loads.

BLOCK DIAGRAM



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.

Electromechanical Switch Replacement

DEBOUNCE.LST

```
1          ;
2          ;                               Debounce
3          ;                               =====
4          ;
5          ;                               by Jim Nagy, June 1997
6          ;
7          ; A program to use the 12C508 as an interface circuit between
8          ; mechanical contacts and electronic circuits. Electronic time delays
9          ; are used to ensure that the positions of up to three switches/contacts
10         ; are stable ('debounced') before passing their status on. No
11         ; external components (other than the switches) are required.
12         ;
13         ; Circuit inputs are GP0(pin7), GP1(pin6), and GP3(pin4),
14         ; while respective outputs are GP5(pin2), GP4(pin3), and GP2(pin5).
15         ; The internal pullups are used for the three inputs, simplifying
16         ; the connection of switches between circuit common and the inputs.
17         ;
18         ; After approximately 0.5sec of inactivity, the '508 puts itself into
19         ; the sleep mode, consuming little power if no switch is being pressed.
20         ; The circuit will wake up (almost instantly) for any change in input.
21         ;
22         ; This circuit inverts the logic level of each input, so a closed switch
23         ; between pin 7 and ground provides a logic high at pin 2, etc. There is
24         ; currently no interaction between the three inputs and the outputs,
25         ; but the logic could easily be altered to provide other functions.
26         ;
27         ; Currently, a 'debounce' time of about 24mS is used, but this can be
28         ; varied (in 2mS increments) by changing the value of DBTime below.
29         ;
30         ; *****
31
32         ;Equates
33         = 0000      W      EQU      0
34         = 0001      F      EQU      1
35         = 0002      Z      EQU      2
36         = 0002      Zero   EQU      2
37         = 0000      C      EQU      0
38         = 0000      Carry  EQU      0
39         = 0007      GPWUF  EQU      7
40
41         = 0000      Sw0    EQU      0      ; switch names
42         = 0001      Sw1    EQU      1
43         = 0003      Sw2    EQU      3
44
45         = 0007      TFlag  EQU      7      ; bit numbers for flags
46
47         = 000C      DBTime EQU      12     ; debounce for 12 * 2.048mS
48
49
50         ; '508 Register Assignments
51         = 0000      INDF   EQU      H'00'
52         = 0001      TMR0   EQU      H'01'
53         = 0002      PCL    EQU      H'02'
54         = 0003      STATUS EQU      H'03'
55         = 0004      FSR    EQU      H'04'
56         = 0005      OSCCAL EQU      H'05'
57         = 0006      GPIO   EQU      H'06'
58
59         ; program variables
60         = 0007      SleepT EQU      H'07' ; inactivity counter to allow sleep
61         = 0008      Flags  EQU      H'08' ; storage space for messages
62         = 0009      ThisSw EQU      H'09' ; current switch status
63
64         ; switch0
```

Electromechanical Switch Replacement

```
65     = 000A     LastSw0 EQU H'0A'      ; last value - for detecting change
66     = 000B     SwState0 EQU H'0B'     ; the official (debounced) status
67     = 000C     DBTimer0 EQU H'0C'     ; counter used during debouncing of the switch
68
69             ; switch1
70     = 000D     LastSw1 EQU H'0D'      ;           "           "
71     = 000E     SwState1 EQU H'0E'     ;           "           "
72     = 000F     DBTimer1 EQU H'0F'     ;           "           "
73
74             ; switch2
75     = 0010     LastSw2 EQU H'10'      ;           "           "
76     = 0011     SwState2 EQU H'11'     ;           "           "
77     = 0012     DBTimer2 EQU H'12'     ;           "           "
78
79
80             ; *****
81             ; Setting the ID words...
82
83             ORG H'0200'
84 0200 0000     ID0           Data.W H'0000'
85 0201 0000     ID1           Data.W H'0000'
86 0202 0000     ID2           Data.W H'0000'
87 0203 0003     ID3           Data.W H'0003'
88
89             ; *****
90             ; and the Fuses...
91             ; MCLRDisabled, NoCodeProtect, NoWatchDog, INTRCOsc
92
93             ORG H'0FFF'
94 0FFF 000A     CONFIG Data.W H'000A'
95
96
97             ; *****
98
99
100            ORG H'00'
101
102 0000 0025     MOVWF  OSCCAL ; store the factory osc. calibration value
103
104 0001 0004     Start  CLRWDT           ; setting up options
105 0002 0C02     MOVLW  B'00000010'     ; use int clock input, /8 prescaler
106 0003 0002     OPTION                    ; pullups on, and wakeup on pin change
107 0004 07E3     BTFSS  STATUS,GPWUF    ; don't change outputs on a wakeup
108 0005 0066     CLRF   GPIO             ; but output 0s on a powerup
109 0006 0C0B     MOVLW  B'00001011'     ; GP0, GP1 and GP3 are inputs,
110 0007 0006     TRIS   GPIO             ; GP2, GP4 and GP5 are outputs
111
112 0008 0067     CLRF   SleepT           ; always start with a clean slate
113 0009 0068     CLRF   Flags
114 000A 006C     CLRF   DBTimer0
115 000B 006F     CLRF   DBTimer1
116 000C 0072     CLRF   DBTimer2
117 000D 06E3     BTFSC  STATUS,GPWUF    ; on wakeup, don't change the switch states
118 000E 0A15     GOTO   Main
119
120 000F 006A     CLRF   LastSw0          ; but if it's a powerup,
121 0010 006B     CLRF   SwState0        ; assume they're all off (open)
122 0011 006D     CLRF   LastSw1
123 0012 006E     CLRF   SwState1
124 0013 0070     CLRF   LastSw2
125 0014 0071     CLRF   SwState2
126
127
128             ; The Main loop has to be executed every 1mS in order to provide
129             ; proper timekeeping, and to monitor the switches often enough.
130             ; In this instance, this is not a problem, but conceivably the loop
```

Electromechanical Switch Replacement

```
131             ; could be expanded to do other things...
132
133 0015 0925      Main   CALL    DoTime      ; need clock 'running'for debounce
134 0016 0934      CALL    Switch0      ; check switch0 position,
135 0017 094D      CALL    Switch1      ; then switch1,
136 0018 0966      CALL    Switch2      ; and switch2
137
138 0019 0040      CLRW                    ; assume no switch is pressed
139
140 001A 022B      MOVF    SwState0,F      ; check the state of sw0
141 001B 0743      BTFSS   STATUS,Zero      ; do nothing if it's not pressed
142 001C 0D20      IORLW   B'00100000'     ; else set GP5 high
143
144 001D 022E      MOVF    SwState1,F      ; same for sw1
145 001E 0743      BTFSS   STATUS,Zero      ; do nothing if it's not pressed
146 001F 0D10      IORLW   B'00010000'     ; output on GP4
147
148 0020 0231      MOVF    SwState2,F      ; and sw2
149 0021 0743      BTFSS   STATUS,Zero      ; do nothing if it's not pressed
150 0022 0D04      IORLW   B'00000100'     ; with output on GP2
151
152 0023 0026      MOVWF   GPIO
153
154 0024 0A15      GOTO    Main
155
156
157
158             ; *****
159             ;           DoTime
160             ; Continually checks TMR0 and updates the program timers
161             ; Returns with W = 0 if there was no time change, and W = 1 if there was.
162
163 0025 07E1      DoTime BTFSS   TMR0,7      ; high bit of timer set?
164 0026 0A29      GOTO    dt1          ; no - we've overflowed
165 0027 05E8      BSF     Flags,TFlag   ; yes, set the flag
166 0028 0800      RETLW  0
167
168 0029 07E8      dt1    BTFSS   Flags,TFlag   ; have the timers been serviced?
169 002A 0800      RETLW  0              ; yes
170
171 002B 02AC      INCF   DBTimer0,F      ; no, increment all the timers
172 002C 02AF      INCF   DBTimer1,F
173 002D 02B2      INCF   DBTimer2,F
174 002E 02A7      INCF   SleepT,F
175 002F 04E8      BCF     Flags,TFlag   ; reset the service flag, and
176 0030 0743      BTFSS   STATUS,Zero      ; see if the sleep timer has overflowed
177 0031 0801      RETLW  1
178
179             ; There's been no activity for 256*2mS, it's time to sleep
180 0032 0206      MOVF   GPIO,W          ; read the current pin status
181 0033 0003      SLEEP                    ; and good night
182
183
184             ; *****
185             ;           Switch0
186             ; Gets the current (debounced) status of the switch connected to GP0 (pin 7)
187             ; Will return with W = 0 if the switch is not pressed and W = 1 if it is.
188
189 0034 0069      Switch0 CLRW      ThisSw      ; assume switch is not pressed
190 0035 0706      BTFSS   GPIO,Sw0      ; check sw status
191 0036 02A9      INCF   ThisSw,F      ; the sw was pressed
192
193             ; Compare to last state
194 0037 020A      MOVF   LastSw0,W
195 0038 0189      XORWF  ThisSw,W
196 0039 0743      BTFSS   STATUS,Zero      ; Zero is set if there's been no change
```

Electromechanical Switch Replacement

```
197 003A 0A46      GOTO    sw01
198
199      ;      No change since last scan, but are we in a debounce phase?
200 003B 020B      MOVF    SwState0,W
201 003C 0189      XORWF  ThisSw,W      ; compare present state of sw to the official state
202 003D 0643      BTFSC  STATUS,Zero ; Zero is Clr if they differ (we're in debounce)
203 003E 0A4A      GOTO    sw02
204
205      ;      Switch is changing, have we gone past the debounce time?
206 003F 0C0C      MOVLW  DBTime      ; get the debounce time
207 0040 008C      SUBWF  DBTimer0,W  ; and compare to elapsed
208 0041 0703      BTFSS  STATUS,Carry ; Carry is set if DBTimer >= DBTicks
209 0042 0A4A      GOTO    sw02
210
211      ;      we've exceeded the debounce time - change the official state of the switch
212 0043 0209      MOVF    ThisSw,W
213 0044 002B      MOVWF  SwState0      ; store current state in SwState
214 0045 0A4A      GOTO    sw02
215
216      ;      the switch input is changing state - prepare to debounce
217 0046 0209      sw01  MOVF    ThisSw,W
218 0047 002A      MOVWF  LastSw0      ; remember this passes' state
219 0048 0067      CLRF   SleepT      ; we've had activity - reset the sleep timer
220 0049 006C      CLRF   DBTimer0    ; and reset the debounce timer
221
222      ;      we're done for this pass...
223 004A 070B      sw02  BTFSS  SwState0,0 ; check the switch state, and
224 004B 0800      RETLW  0            ; return with 0 if not pressed,
225 004C 0801      RETLW  1            ; 1 if pressed
226
227
228      ;      *****
229      ;      Switch1
230      ;      Gets the current (debounced) status of the switch connected to GP1 (pin 6)
231      ;      Will return with W = 0 if the switch is not pressed and W = 1 if it is.
232
233 004D 0069      Switch1 CLRF   ThisSw      ; assume switch is not pressed
234 004E 0726      BTFSS  GPIO,Sw1      ; check sw status
235 004F 02A9      INCF   ThisSw,F      ; the sw was pressed
236
237      ;      Compare to last state
238 0050 020D      MOVF    LastSw1,W
239 0051 0189      XORWF  ThisSw,W
240 0052 0743      BTFSS  STATUS,Zero   ; Zero is set if there's been no change
241 0053 0A5F      GOTO    sw11
242
243      ;      No change since last scan, but are we in a debounce phase?
244 0054 020E      MOVF    SwState1,W
245 0055 0189      XORWF  ThisSw,W      ; compare present state of sw to the official state
246 0056 0643      BTFSC  STATUS,Zero ; Zero is Clr if they differ (we're in debounce)
247 0057 0A63      GOTO    sw12
248
249      ;      Switch is changing, have we gone past the debounce time?
250 0058 0C0C      MOVLW  DBTime      ; get the debounce time
251 0059 008F      SUBWF  DBTimer1,W  ; and compare to elapsed
252 005A 0703      BTFSS  STATUS,Carry ; Carry is set if DBTimer >= DBTicks
253 005B 0A63      GOTO    sw12
254
255      ;      we've exceeded the debounce time - change the official state of the switch
256 005C 0209      MOVF    ThisSw,W
257 005D 002E      MOVWF  SwState1      ; store current state in SwState
258 005E 0A63      GOTO    sw12
259
260      ;      the switch input is changing state - prepare to debounce
261 005F 0209      sw11  MOVF    ThisSw,W
262 0060 002D      MOVWF  LastSw1      ; remember this passes' state
```

Electromechanical Switch Replacement

```
263 0061 0067          CLRF   SleepT      ; we've had activity - reset the sleep timer
264 0062 006F          CLRF   DBTimer1    ; and reset the debounce timer
265
266          ;          we're done for this pass...
267 0063 070E          sw12   BTFSS   SwState1,0  ; check the switch state, and
268 0064 0800          RETLW  0              ; return with 0 if not pressed,
269 0065 0801          RETLW  1              ; 1 if pressed
270
271
272          ;          *****
273          ;          Switch2
274          ;          Gets the current (debounced) status of the switch connected to GP3 (pin 4)
275          ;          Will return with W = 0 if the switch is not pressed and W = 1 if it is.
276
277 0066 0069          Switch2 CLRF   ThisSw    ; assume switch is not pressed
278 0067 0766          BTFSS  GPIO,Sw2      ; check sw status
279 0068 02A9          INCF   ThisSw,F      ; the sw was pressed
280
281          ;          Compare to last state
282 0069 0210          MOVF   LastSw2,W
283 006A 0189          XORWF  ThisSw,W
284 006B 0743          BTFSS  STATUS,Zero    ; Zero is set if there's been no change
285 006C 0A78          GOTO   sw21
286
287          ;          No change since last scan, but are we in a debounce phase?
288 006D 0211          MOVF   SwState2,W
289 006E 0189          XORWF  ThisSw,W      ; compare present state of sw to the official state
290 006F 0643          BTFSC  STATUS,Zero   ; Zero is Clr if they differ (we're in debounce)
291 0070 0A7C          GOTO   sw22
292
293          ;          Switch is changing, have we gone past the debounce time?
294 0071 0C0C          MOVLW  DBTime        ; get the debounce time
295 0072 0092          SUBWF  DBTimer2,W    ; and compare to elapsed
296 0073 0703          BTFSS  STATUS,Carry  ; Carry is set if DBTimer >= DBTicks
297 0074 0A7C          GOTO   sw22
298
299          ;          we've exceeded the debounce time - change the official state of the switch
300 0075 0209          MOVF   ThisSw,W
301 0076 0031          MOVWF  SwState2      ; store current state in SwState
302 0077 0A7C          GOTO   sw22
303
304          ;          the switch input is changing state - prepare to debounce
305 0078 0209          sw21   MOVF   ThisSw,W
306 0079 0030          MOVWF  LastSw2      ; remember this passes' state
307 007A 0067          CLRF   SleepT      ; we've had activity - reset the sleep timer
308 007B 0072          CLRF   DBTimer2    ; and reset the debounce timer
309
310          ;          we're done for this pass...
311 007C 0711          sw22   BTFSS   SwState2,0  ; check the switch state, and
312 007D 0800          RETLW  0              ; return with 0 if not pressed,
313 007E 0801          RETLW  1              ; 1 if pressed
314
315          END
```