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.

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

The BIL TJR 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 PIC 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 BIL TJR was a PIC16C84-based circuit which will not be described in detail; however, its schematic is enclosed. The 84-based circuit is a superset 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 time-keeping 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 diagram 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.
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)
GRAPHICAL HARDWARE REPRESENTATION

BRIGHT IDEA LIGHT TIMER JR.

CONNECT PIN 4 OF DS1202 TO GROUND, LEAVE PIN 1 UNCONNECTED, CONNECT 0.1uF CAPACITOR BETWEEN PINS 8 AND 4 FOR TESTING PURPOSES.

LED AND RESISTOR FOR TESTING PURPOSES.

TO PC PARALLEL PORT FOR PROGRAMMING DS1202.

BATTERY BACKED RAM (REMOVE FOR NORMAL OPERATION).

CONNECT 0.1uF CAPACITOR BETWEEN PINS 14 AND 5 ON PIC16C84 FOR FUTURE EXPANSION.

C3 .1uF
C1 .1uF
C2 .1uF
C4 .1uF

U1 MAX202E

C2+ C2- C1+ C1- V+ V- 4 5 1 3 13 14 8 7 10 9 11 12 2 6

R1 10 K

D1 LED

Y1 4 MHz

R2 1K

S1

C5 33 pF

U2 PIC16C84

RA21 RA32 RA4/T0CKI3 MCLR/Vpp 4 Vss/GND 5 RB06 RB17 RB28 RB39 RB4 10 RB5 11 RB6/CLOCK 12 RB7/DATA 13 Vdd/+5V 14 OSC2/CLKOUT 15 OSC1/CLKIN 16 RA0 17 RA1 18

JP1 J1

BRIGHT IDEA LIGHT TIMER JR. (TEST BED)

CONNECT PIN 4 OF DS1202 TO GROUND, LEAVE PIN 1 UNCONNECTED, CONNECT 0.1uF CAPACITOR BETWEEN PINS 8 AND 4 FOR TESTING PURPOSES.

LED AND RESISTOR FOR TESTING PURPOSES.

TO PC PARALLEL PORT FOR PROGRAMMING DS1202.

BATTERY BACKED RAM (REMOVE FOR NORMAL OPERATION).

CONNECT 0.1uF CAPACITOR BETWEEN PINS 14 AND 5 ON PIC16C84 FOR FUTURE EXPANSION.

C3 .1uF
C1 .1uF
C2 .1uF
C4 .1uF

U1 MAX202E

C2+ C2- C1+ C1- V+ V- 4 5 1 3 13 14 8 7 10 9 11 12 2 6

R1 10 K

D1 LED

Y1 4 MHz

R2 1K

S1

C5 33 pF

U2 PIC16C84

RA21 RA32 RA4/T0CKI3 MCLR/Vpp 4 Vss/GND 5 RB06 RB17 RB28 RB39 RB4 10 RB5 11 RB6/CLOCK 12 RB7/DATA 13 Vdd/+5V 14 OSC2/CLKOUT 15 OSC1/CLKIN 16 RA0 17 RA1 18

JP1 J1

BRIGHT IDEA LIGHT TIMER JR. (TEST BED)

CONNECT PIN 4 OF DS1202 TO GROUND, LEAVE PIN 1 UNCONNECTED, CONNECT 0.1uF CAPACITOR BETWEEN PINS 8 AND 4 FOR TESTING PURPOSES.

LED AND RESISTOR FOR TESTING PURPOSES.

TO PC PARALLEL PORT FOR PROGRAMMING DS1202.

BATTERY BACKED RAM (REMOVE FOR NORMAL OPERATION).

CONNECT 0.1uF CAPACITOR BETWEEN PINS 14 AND 5 ON PIC16C84 FOR FUTURE EXPANSION.

C3 .1uF
C1 .1uF
C2 .1uF
C4 .1uF

U1 MAX202E

C2+ C2- C1+ C1- V+ V- 4 5 1 3 13 14 8 7 10 9 11 12 2 6

R1 10 K

D1 LED

Y1 4 MHz

R2 1K

S1
APPENDIX A: SOURCE CODE

;comment out one or the other of the 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
#include "p12c508.inc" ; __12C508
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 sublw sublw ; fix for microchip's bad mnemonic
#define SUBWL sublw ; both upper and lower case

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

SKPGEZ macro
btfss STATUS,C
endm ; used after a subtract instruction, this macro
; will skip the next instruction if the result
; 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);
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, SKPNZ, |
;| SKPDC, SKPNDC, SKPZ, SKPNZ, TSTF, MOVFW            |
;\(can use in upper or lower case)\n
;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
endc
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 OVRIDE_ 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)
#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, disable 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 RAMNCLK bitVars,0       ;access DS1202 RAM/not clock indicator
#define LITESHD bitVars,1        ;output on/off shadow bit

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

org RESET ;reset sends execution here
ifdef __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                                                            |
;\--------------------------------------------------------------------------/
ifdef __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
defc 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 5 + 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
SRPNC goto send1L1
goto send0
start goto 5 + 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
if (high jmpStrt != high jmpEnd)
error  "jump table crosses page boundary in subroutine send232!"
endif

endif ;__16C84

;if (high jmpStrt != high jmpEnd)
error  "jump table crosses page boundary in subroutine send232!"
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                    ;__12C508
 movlw   DATAOUT
 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
 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
 bcf     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
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

deftsz 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

deftsz jay,f ; skip next if done getting all data bytes
    goto rdLoop2 ; continue 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
};--------------------------------------------------------------------------/

ifdef __16C84

wrClock bcf 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

deftsz 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         ;continue 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
else                    ;__12C508
movlw   DATAINP
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

MOVFWF  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

r1202lp 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
decfsz jay, f       ;skip next if clocked in all 8 command bits
goto r1202lp        ;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
r1202l2 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   dat1202, f      ;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 r1202l2         ;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       |
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 RAMNCLK ;skip next if really writing to RAM area
bcf temp,RAM1202 ;really writing to clock area of DS1202
bsf STATUS,RP0 ;make the data i/o pin an output temporarily
bcf TRISB,IOPIN
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
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
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,f
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
movwf 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

endif

© 1997 Microchip Technology Inc.
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
  bcf     OPTION_REG,T0CS ;select external source for timer 0
  bcf     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   TRISB
  bcf     STATUS,RP0      ;set default page back to 0
else                    ;__12C508
  clrf    GPIO           ;set port output latches to a known state
endif
movlw   OPTREG
option                  ;disable wake-up, disable pull-ups, 1:1 watchdog
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

endif
movlw  h'96'
movwf  yr1202

;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
bcf RAMNCLK

;disallow writes to DS1202

call wr1202

;DEBUG

endif

;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 bc2d2asc

call send232

MOVFW hr1202

call bc2d2asc

call send232

movlw ':'

call send232

MOVFW min1202

call bc2d2asc

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

xorf 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
rdNxChg MOVFW dow1202 ;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 adrl202 ;store index as RAM location to read in DS1202
bsf RAMNCLK
call rd1202 ;read DS1202 RAM location [day * 4 + state]
MOVFW datl202 ;retrieve HHHHHTTT binary data
movwf temp ;save it for rotating
rrf temp,f ;rotate to get CHHHHHTT
rrf temp,f ;rotate to get CCHHHHHT
rrf temp,w ;rotate to get CCHHHHHT
andlw b'00001111' ;mask to get 0000HHHH
call bin2bcd ;convert to 2 BCD digits
rlf bcdl,f ;rotate MS BCD digit to get 0000MMMC
rlf bcdl,f ;rotate MS BCD digit to get 0000MMMC
rlf bcdl,f ;rotate MS BCD digit to get 0000MMMC
rlf bcdl,w ;rotate MS BCD digit to get MMMMMMCC
andlw b'11110000' ;mask to get MMMM0000
iorwf bcdl,w ;combine to get MMMMLLLL hours
movwf chgHrs ;save for comparison to current hours later
clf temp ;clear the addition accumulator
MOVFW datl202 ;retrieve HHHHHTTT binary data
andlw b'00000111' ;mask out hours to get 00000TTT
movlw 10 ;add 10 to accumulated sum each time thru
addLoop TSTF eye ;i down to 0 yet?
SKPNZ goto addDone ;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 bcdl,f ;rotate MS BCD digit to get 0000MMMC
rlf bcdl,f ;rotate MS BCD digit to get 0000MMMC
rlf bcdl,f ;rotate MS BCD digit to get 0000MMMC
rlf bcdl,w ;rotate MS BCD digit to get MMMMMMCC
andlw b'11110000' ;mask to get MMMM0000
iorwf bcdl,w ;combine to get MMMMLLLL 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 ;time is after 2nd turn off time, recycle
chkTime MOVFW chgHrs
subwf hr1202,w ;w = current hours - next change hours
SKPGEZ goto updShdw ;skip next if current >= next change
SKPZ goto 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
goto updShdw ;go update output from shadow bit
Electromechanical Switch Replacement

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

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

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

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

end
NOTES: