

```

;*****
; Program      :   PWMETER.ASM
; Function     :   RC Pulse Width Meter displaying on 16x2 LCD
; Author       :   John F. Fitter B.E.
; Language     :   PIC Assembler
; Platform     :   PIC 16C84 @ 16MHz
; Revisions    :   27jul96 Original
;              :   Source reformatted 3mar99
;
; Copyright c 1996 Eagle Air Australia Pty. Ltd. All rights reserved
;*****

HARD_PN0 equ 0 ; Hardware part number first field
HARD_PN1 equ 0 ; Hardware part number second field
SOFT_VER equ 1 ; Software version number
SOFT_REV equ 0 ; Software revision number
SOFT_BMO equ 7 ; Software build month
SOFT_BYR equ 96 ; Software build year
SPARE_0 equ 0 ; <spare field>
SPARE_1 equ 0 ; <spare field>

list p=16C84, r=dec, b=6, x=OFF
CPU equ 1684
CLK equ 16 ; clock speed in MHz (16, 8, or 4 only)
SIM equ 0 ; Change timing constants for simulator
include "e16c84.inc" ; include enhanced defines for 16C84

__config __CP_OFF & __PWRTE_ON & __WDT_ON & __HS_OSC ; configuration fuses

;*****
; LCD equates
;*****

LCDPORT equ PORTB ; lcd is on port b
LCDTRIS equ TRISB

SET_8B equ 0x30 ; set 8 bit data (for initialisation)
SET_4B_1L equ 0x20 ; set 4 bit data, 1 lines
SET_4B_2L equ 0x28 ; set 4 bit data, 2 lines

DISP_ON_C equ 0x0E ; display ON, cursor ON, blink OFF
DISP_ON equ 0x0C ; display ON, cursor OFF, blink OFF
DISP_ON_CB equ 0x0F ; display ON, cursor ON, blink ON
DISP_ON_B equ 0x0D ; display ON, cursor OFF, blink ON
DISP_OFF equ 0x08 ; display OFF
CLR_DISP equ 0x01 ; clear display
HOME_DISP equ 0x02 ; home display
SHR_C equ 0x1C ; shift right 1 char
SHL_C equ 0x18 ; shift left 1 char
ENTRY_INC equ 0x06 ; entry mode increment, no shift
ENTRY_INC_S equ 0x07 ; entry mode increment, shift
ENTRY_DEC equ 0x04 ; entry mode decrement
ENTRY_DEC_S equ 0x05 ; entry mode decrement, shift
DD_RAM_ADDR equ 0x80 ; least significant 7 bits are for address
DD_RAM_UL equ 0x80 ; upper left corner of display

STRT_LINE1 equ 0 ; parameter to set address to line 1 start
STRT_LINE2 equ 0x40 ; parameter to set address to line 2 start

;*****
; Register assignments
;*****

TEMP1 equ 0x10 ; temporary register 1
TEMP2 equ 0x11 ; temporary register 2
TEMP3 equ 0x12 ; temporary register 3
TOFFS equ 0x13 ; string table offset holding register

DEL0 equ 0x14 ; delay register0
DEL1 equ 0x15 ; delay register1
DEL2 equ 0x16 ; delay register2
DEL3 equ 0x17 ; delay register3

T1HI equ 0x18 ; timer count 1 high byte
T1MI equ 0x19 ; timer count 1 middle byte
T1LO equ 0x1A ; timer count 1 low byte
T2HI equ 0x1B ; timer count 2 high byte

```

```

T2MI      equ    0x1C      ; timer count 2 middle byte
T2LO      equ    0x1D      ; timer count 2 low byte

HIBYTE    equ    0x1E      ; 16 bit binary high byte
LOBYTE    equ    0x1F      ; 16 bit binay low byte

BCD0      equ    0x20      ; bcd msd (rh nibble)
BCD1      equ    0x21      ; bcd digits 2 & 3
BCD2      equ    0x22      ; bcd digits 0 & 1

TIMERH    equ    0x23      ; timer high byte (count of overflows)
TIMERM    equ    0x24      ; timer middle byte (count of overflows)

KEY_CTR   equ    0x25      ; count of button senses ignored
PLS_CTR   equ    0x25      ; count of input pulses ignored

FLAGS     equ    0x26      ; system status flags register
OUTPULSE  equ    0x27      ; output pulse width (pw - 500 us) / 10
NEUTPULSE equ    0x28      ; neutral pulse width working register

;*****
; Port definitions
;*****

LCDWRITE  equ    B'10000000' ; LCDTRIS register for writes
LCDREAD   equ    B'10001111' ; LCDTRIS register for reads

; LCD Port pin assignments
bit LCD_DB4, 0, LCDPORT ; LCD data bit 4
bit LCD_DB5, 1, LCDPORT ; LCD data bit 5
bit LCD_DB6, 2, LCDPORT ; LCD data bit 6
bit LCD_DB7, 3, LCDPORT ; LCD data bit 7
bit LCD_RW, 4, LCDPORT ; LCD read/write
bit LCD_RS, 5, LCDPORT ; LCD register select
bit LCD_E, 6, LCDPORT ; LCD data bus clock
bit SENSE, 7, LCDPORT ; Switch sense line
bit SENSE_DIR, 7, LCDTRIS ; Switch sense line direction bit

PORTATRIS equ    B'11110111' ; Port A initial i/o directions
; and pin assignments
bit MODE, 0, PORTA ; Mode button
bit MODE_DIR, 0, TRISA ; Mode button direction bit
bit UPSCAN, 1, PORTA ; Up button
bit UPS_DIR, 1, TRISA ; Up button direction bit
bit DNSCAN, 2, PORTA ; Down button
bit DNS_DIR, 2, TRISA ; Down button direction bit
bit SERVO, 3, PORTA ; Servo signal output pin
bit PULSE, 4, PORTA ; Pulse input pin

INIFLAGS  equ    0 ; Initial system flags status
SCRFLAGMASK equ    B'11111100' ; Flags mask to clear screen drawn bits

bit FLG_MIDS, 0, FLAGS ; main input data display screen is drawn
bit FLG_NSIDS, 1, FLAGS ; no signal display screen is drawn
bit FLG_UP, 2, FLAGS ; up button pressed
bit FLG_DN, 3, FLAGS ; down button pressed
bit FLG_MODE, 4, FLAGS ; mode button pressed
bit PLS_STAT, 5, FLAGS ; saved status of pulse input pin
bit FLG_6, 6, FLAGS ; unused
bit FLG_7, 7, FLAGS ; unused
PLS_STATBIT equ    5 ; must be same as status above

;*****
; General Equates
;*****

INOPTION  equ    B'10001101' ; Initial option register set
; port b pullups disabled
; clock in timer mode
; prescaler assigned to wdt, rate = 32

ININTCON  equ    B'00100000' ; Initial interrupt control register set
; all interrupts masked

NOPLS    equ    20 ; number of initial input pulses to
; ignore redirecting to output

```

```

;*****
; EEPROM Data
;*****

NEUTRPLS    org      0x2100
            de        100                ; default (neutral pulse width - 500) / 10

;*****
; Macros and assembler variables
;*****

; Timeout is determined as follows for a timeout period of 100000 us :-
; The timeout counter is 14 cycles in the minor loop and 10 cycles in the major loop which is
; 10 + 10/256 cycles per loop. Loop count is multiplied by the clock speed multiples of 4MHz
; (1us cycle @ 4MHz). Loop ctr is an up-counter so loop count must be subtracted from 0xffff

TIMEOUT_L   equ      low  (0xFFFF - (((100000 / 599) * 256) / 6) * (CLK >> 2))
TIMEOUT_H   equ      high (0xFFFF - (((100000 / 599) * 256) / 6) * (CLK >> 2))

variable    count

tgl_clk     macro
            seb      LCD_E                ; toggle lcd data bus clock
            if CLK > 10
            nop                        ; need delay for more than 10MHz
            endif
            clb      LCD_E                ; min PWeh=220ns, Tcy=500ns
            endm

; Macro to write a page to lcd where a page consists of two lines of text.
; Each line consists of a null terminated string of characters beginning at _string1 for the
; first line and _string2 for the second line.

wrt_page    macro    _string1, _string2
            movlw    STRT_LINE1
            call     lcd_sladdr
            movlw    _string1 - str0
            call     lcd_str
            movlw    STRT_LINE2
            call     lcd_sladdr
            movlw    _string2 - str0
            call     lcd_str
            andlf    FLAGS, SCRFLAGMASK
            endm

; Macro to clear the display of the lcd.

clear_disp  macro
            movlw    CLR_DISP
            call     lcd_cmd
            endm

; Macro to shift 3 registers right by _n bits. The register addresses are _r1, _r2 and _r3

shrt3      macro    _n, _r1, _r2, _r3
            local   count = _n
            while  count > 0
            clb     B_C
            rrf     _r1
            rrf     _r2
            rrf     _r3
            count--
            endw
            endm

; Macro to shift 3 registers left _n bits. The register addresses are _r1, _r2 and _r3

shlft3     macro    _n, _r1, _r2, _r3
            local   count = _n
            while  count > 0
            clb     B_C
            rlf     _r3
            rlf     _r2
            rlf     _r1
            count--
            endw
            endm

```

```
; Macro to perform 24 bit addition. The numbers to be added are in srch, srcm and srcl
; representing the high byte, middle byte, and low bytes respectively. The other number to be
; added is in desth, destm, and destl and the summ is placed in these registers.
; The carry flag does not represent the result of the addition.
```

```
add24      macro    srch, srcm, srcl, desth, destm, destl
            local   _add24a, _add24b
            movfw   srcl
            addwf   destl
            bnc     _add24a
            movlw   1
            addwf   destm
            bnc     _add24a
            movlw   1
            addwf   desth
_add24a    movfw   srcm
            addwf   destm
            bnc     _add24b
            movlw   1
            addwf   desth
_add24b    movfw   srch
            addwf   desth
            endm
```

```
; Macro to clear a 24 bit register. The 24 bit number to be reset is in the three registers
; reghi, regmi, and reglo.
```

```
clr24      macro    reghi, regmi, reglo
            clrfs   reghi
            clrfs   regmi
            clrfs   reglo
            endm
```

```
; Macro to move 24 bits from src to dest. The source registers are srch, srcm, and srcl.
; A copy of this number is placed in desth, destm, and destl.
```

```
mov24      macro    srch, srcm, srcl, desth, destm, destl
            movff   srch, desth
            movff   srcm, destm
            movff   srcl, destl
            endm
```

```
; Macro to write 4 bcd digits to the lcd on specified line to specified precision. The packed
; bcd number to be written to the lcd is in _bcd1 and _bcd2 with the most significant digit in
; the high nibble of _bcd1. _line is the line number to write to on the lcd (1 or 2). _pos is
; the character position of the start of the number string (0 to 39). _prec specifies the
; location of the decimal point in character positions from the end of the string, ie. 2 for
; hundredths.
```

```
lcd4bcd    macro    _line, _pos, _bcd1, _bcd2, _prec
            movlw   ((_line-1)<<6)|_pos      ; set lcd address to line and char pos
            call    lcd_sladdr
            swapf   _bcd1, W                ; get digit 3 into W
            call    lcd_hex
            if _prec == 3
            movlw   '.'                      ; write the dp
            call    lcd_char
            endif
            movfw   _bcd1                    ; get digit 2 into W
            call    lcd_hex
            if _prec == 2
            movlw   '.'                      ; write the dp
            call    lcd_char
            endif
            swapf   _bcd2, W                ; get digit 1 into W
            call    lcd_hex
            if _prec == 1
            movlw   '.'                      ; write the dp
            call    lcd_char
            endif
            movfw   _bcd2                    ; get digit 0 into W
            call    lcd_hex
            endm
```

```
; Macro to delay for ms milliseconds.
```

```

delay    macro    ms
          if ms == 0
          nop                                ; +1 cycle error
          else
          if ms == 1
          movlw  100 + 4 / CLK                ; no error at 4 MHz
          call   udelay                       ; - 10 cycle error at higher speeds
          else
          if ms < 256
          movlw  ms                           ; -4 cycle error
          call   sdelay
          else
          movlw  ms / 100                     ; +2 cycle error
          call   ldelay
          endif
          endif
          endif
          endm

;*****
; START OF CODE
;*****

;*****
; Interrupt service routine
; Only one interrupt source is specified for this program so it must be from the timer. The
; isr can reside at the interrupt vector (only one isr)
; The isr does not effect w or status so context saving is not necessary
;*****

intvec    org      _intvec                    ; only one interrupt - must be from timer
          clb      B_TOIF                    ; clear the interrupt flag
          incfsz   TIMERM                    ; increment the timer middle byte
          retfie   ; no overflow, then return
          incfsz   TIMERH                    ; if overflow, increment high byte
          nop
          clrwdt   ; good place to clear the watchdog timer
          retfie   ; return and enable interrupts

;*****
; Power-on reset entry
;*****

init      clrf     STATUS                    ; power-on reset entry
          clrf     PORTA                    ; clear the port latches
          clrf     PORTB
          movlf    INTCON, ININTCON         ; set the interrupt control register
          movlf    FLAGS, INIFLAGS        ; set the system flags register
          call     rdnpw                    ; get default neutral output pulse width
          movwf    OUTPULSE                ; from eeprom
          movwf    NEUTPULSE
          seb      B_RP0                    ; select register bank 1
          movlf    OPTION_REG, INOPTION    ; set the options register
          movlf    TRISB, 0                ; set all port b pins to outputs
          movlf    TRISA, PORTATRIS       ; setup port a pins
          clb      B_RP0                    ; select register bank 0
          call     lcd_init                 ; initialize the lcd
          wrt_page str1, str2              ; write power up messages to lcd
          delay    3000                    ; 3 second delay
          call     lcd_shl16                ; shift display left 16 char
          delay    1500                    ; 1.5 second delay

;*****
; Main pulse input program loop
; The main loop is to _main0. main_inp is the return point from the generating pulse mode
; of operation. The different entries are to allow for the suppression of 10 pulses to
; eliminate problems with noisy input lines.
;*****

main_inp  movlf    PLS_CTR, 2*NOPLS        ; do not send the first x pulses to output
_main0    clr24    BCD2, BCD1, BCD0        ; clear summation registers
          clr24    TEMP3, TEMP2, TEMP1
          seb      B_GIE                    ; enable the overflow interrupt
          movlf    TOFFS, 8                ; counter to get 8 pulses and add together

; The input pulse times are added for 8 pulses and the sum divided by 8 (ie. averaged).
; This method is a first approach to noise elimination. If no signal is found during the pulse

```

```

; edge timeout period then it is assumed that there is no signal and the operating mode is
; changed to that of producing pulses.
; destination = TEMP 1, 2, & 3 for T1
;               BCD 0, 1, & 2 for T2

_main1    call    gpet                ; get the pulse edge times
          bnc     _main3              ; carry set means timeout occurred
          bbs     FLG_NSDFS, _main2   ; is screen already drawn ?
          clear_disp                 ; no, then clear display
          wrt_page str5, str6         ; and draw no signal screen
          seb     FLG_NSDFS           ; and set the no-signal screen drawn flag
_main2    delay   200                 ; wait for 200 ms
          goto    main_out            ; no input signal, then make output signals
          ; valid signal, update the summation registers

_main3    add24   T2HI, T2MI, T2LO, BCD2, BCD1, BCD0
          add24   T1HI, T1MI, T1LO, TEMP3, TEMP2, TEMP1
          decfsz  TOFFS
          goto    _main1              ; loop again until done 8 times

; Get 24 bit sums back into timer registers. These are divided by 8 which will yield an 8
; sample mean (right shift by 3 is divide by 8).

          mov24   BCD2, BCD1, BCD0, T2HI, T2MI, T2LO
          mov24   TEMP3, TEMP2, TEMP1, T1HI, T1MI, T1LO

          bbs     FLG_MIDS, _main4     ; is screen already drawn ?
          clear_disp                 ; no, then clear display
          wrt_page str3, str4         ; and draw main input data display screen
          seb     FLG_MIDS            ; and set the main input screen drawn flag

; Now divide the timers by 8 and a further division to account for the speed of the
; processor clock (4/16MHz, 2/8MHz, 1/4MHz).

_main4    shrt3   (CLK >> 3) + 3, T2HI, T2MI, T2LO
          shrt3   (CLK >> 3) + 3, T1HI, T1MI, T1LO

; Compute the frame width, fw and leave the result in the bcd conversion registers, do the bcd
; conversion, then divide by 10 (4 bit right shift) because the frame width is only displayed
; to a resolution of 10us. The result is then displayed.

          movff   T2LO, LOBYTE
          movff   T2MI, HIBYTE
          call    b2_bcd
          shrt3   4, BCD0, BCD1, BCD2
          lcd4bcd 2, 8, BCD1, BCD2, 2

; Compute the pulse width, pw by a 24 bit subtraction of t1 from t2. The assumption is that
; t2 >= t1 and the difference is not more than 16 bits. The result is left in the bcd
; conversion registers, the conversion done and the result displayed.

          movfw   T1LO                ; 24 bit to 16 bit subtraction
          subwf   T2LO, W
          bc      _main5              ; and put the result in the bcd conversion
          incf    T1MI                 ; registers
_main5    movwf   LOBYTE               ; and convert pw to bcd
          movfw   T1MI
          subwf   T2MI, W
          movwf   HIBYTE
          call    b2_bcd
          lcd4bcd 1, 8, BCD1, BCD2, 3

          goto    _main0              ; end of main input program loop

;*****
; Main pulse output program loop
;*****

main_out  clb     B_GIE                ; turn off interrupts
          clb     SERVO                 ; ensure servo signal line is low to start
          movlf   KEY_CTR, 10           ; set count of button senses to skip

; The input signal is checked every cycle through this code to identify if the input signal
; line has changed state. To do this the signal line status on entry is saved.

          clb     PLS_STAT              ; save the status of the pulse input pin
          skbc   PULSE
          seb     PLS_STAT

```

```
; Compare status of pulse input pin to the saved status - if different and meeting the criteria
; for a stable difference then there is a signal on the line so goto the main input code. This
; is the start of the main loop so this check is done for each pulse generated.
```

```
_mout0      bbc      PLS_STAT, _mout2
```

```
; The line was high. It must now stay low for 200us to be a valid input signal.
```

```
_mout1      movlf    TEMP1, CLK * 10      ; so check that it stays low for 200us
            bbs     PULSE, _mout4      ; line has gone/is still high so fail check
            decfsz  TEMP1
            goto    _mout1
            goto    main_inp          ; on loop exit, signal check has succeeded
```

```
; The line was low. It must now stay high for 200us to be a valid input signal.
```

```
_mout2      movlf    TEMP1, CLK * 10      ; check that it stays high for 200us
_mout3      bbc     PULSE, _mout4      ; line has gone/is still low so fail check
            decfsz  TEMP1
            goto    _mout3
            goto    main_inp          ; on loop exit, signal check has succeeded
```

```
; If we get to here then there is no input signal so we remain in the pulse generation mode of
; operation. Prior to generating a pulse, check for and process any user input. There are only
; three buttons and their function is as follows;
```

```
; Scan up           Pulse width increases by 10us while button is held
; Scan down        Pulse width decreases by 10us while button is held
; Scan up + Fast   Pulse width increases 10 times as fast
; Scan down + Fast Pulse width decreases 10 times as fast
; Scan up + Scan down Pulse width returns to saved neutral pulse width
; Fast (held for 1.5 sec) Current pulse width saved as neutral pulse width
```

```
; The fast/slow logic is simple. Normally, the user input is only scanned once every 10 cycles
; through the loop (10 output pulses). When the fast button is pressed, scanning is done every
; cycle.
```

```
; The pulse width maths is as follows; The output pulse width is equal to 500 us plus 10 x the
; value of an 8 bit number, in this case stored in OUTPUTPULSE.
```

```
; The neutral pulse width is stored in eeprom and can be updated by the user.
```

```
_mout4      call     getbuttons          ; get any user input
            decf    KEY_CTR             ; only want to scan every 10th pulse
            bnz     _mout10
            movlf   KEY_CTR, 10        ; reset count of button senses to skip

            bbc     FLG_UP, _mout7     ; is the up button pressed ?
            bbc     FLG_DN, _mout6     ; yes, then are both buttons pressed ?
            movff   NEUTPULSE, OUTPUTPULSE ; yes, then set neutral pulse width
            goto    _mout10

_mout6      movlw   1                  ; up button is pressed
            skbc    FLG_MODE           ; then, is mode button pressed ?
            movwf   KEY_CTR           ; yes, then scan on every pulse
            incf    OUTPUTPULSE       ; and scan up by 10us
            goto    _mout8

_mout7      bbc     FLG_DN, _mout7a    ; not up button, then is the down button pressed ?
            movlw   1                  ; down button is pressed
            skbc    FLG_MODE           ; then, is mode button pressed ?
            movwf   KEY_CTR           ; yes, then scan on every pulse
            decf    OUTPUTPULSE       ; and scan down by 10us
            goto    _mout8

_mout7a     bbc     FLG_MODE, _mout8   ; not up or down, then is mode button pressed ?
_mout7b     movlf   TEMP1, 150         ; yes, then check if mode button ONLY pressed
            call    getbuttons         ; for 1500 ms. If so then save current output
            bbs     FLG_UP, _mout8     ; pulse width as new neutral pulse width.
            bbs     FLG_DN, _mout8
            bbc     FLG_MODE, _mout8   ; up or down pressed or mode released and quit
            delay   10                 ; delay for 10 ms
            decfsz  TEMP1
            goto    _mout7b           ; loop to check mode button again

            movfw   OUTPUTPULSE       ; mode button has been down for 1500 ms
            movwf   NEUTPULSE         ; set new neutral pulse width
            call    wrnpw             ; and save in eeprom
```

```

clear_disp          ; clear display
wrt_page str7, str8 ; draw setting neutral pulse screen
movff  NEUTPULSE, LOBYTE ; get neutral pw into bcd conversion regs
clrf  HIBYTE
addlf  LOBYTE, 50      ; 16 bit add 50 to it
skpnc
incf  HIBYTE

call  b2_bcd          ; convert pw to bcd
shlft3 4, BCD0, BCD1, BCD2 ; multiply by 10
lcd4bcd 2, 8, BCD1, BCD2, 3 ; display on lcd
delay 2000           ; display for 2 seconds
call  lcd_shl16      ; shift display left 16 char
clear_disp          ; clear display
wrt_page str5, str6 ; and draw no signal screen

; Truncate the output pulse width to 500 us either side of neutral to avoid accidentally
; driving the servo to the stops.

_mout8  movlw 50          ; limit operation to NEUTPULSE +/- 500 us
        addwf NEUTPULSE, W
        subwf OUTPULSE, W ; is it more than NEUTPULSE + 50 ?
        bnc  _mout9
        movlw 50          ; yes, set to NEUTPULSE + 50
        addwf NEUTPULSE, W
        movwf OUTPULSE
        goto _mout10
_mout9  movlw 50
        subwf NEUTPULSE, W
        subwf OUTPULSE, W ; is it less than NEUTPULSE - 50
        bc  _mout10
        movlw 50          ; yes, set to NEUTPULSE - 50
        subwf NEUTPULSE, W
        movwf OUTPULSE

; Send the output pulse to the servo - this is what it's all about

_mout10 movfw OUTPULSE
        call  mkpls

; The output pulse width is now converted to bcd, multiplied by 10, and displayed on the lcd.

movff  OUTPULSE, LOBYTE ; get output pw into bcd conversion regs
clrf  HIBYTE
addlf  LOBYTE, 50      ; 16 bit add 50 to it
skpnc
incf  HIBYTE
call  b2_bcd          ; convert pw to bcd
shlft3 4, BCD0, BCD1, BCD2 ; multiply by 10
lcd4bcd 2, 8, BCD1, BCD2, 3 ; display on lcd

; Delay to establish the frame width. Although the servos are not critical of frame width, we
; will attempt to ensure about 20 ms. All of the above code takes about 4 ms depending on which
; buttons are pressed and how much noise is on the input line.

delay 16             ; 16 ms delay
goto  _mout0        ; loop and check if input line has changed

;*****
; Subroutine to generate a servo pulse
; The width of the pulse is 500 us + 10W us
;*****

mkpls  seb  SERVO          ; send start of servo pulse
        call  udelay       ; delay for 10W - 10 cycles
        movlw 50
        call  udelay       ; delay for 500 us - 10 cycles
        movlf TEMP1, 5
_mkpls0 decfsz TEMP1      ; adjust for latency (20 - 4 = 16 cycles)
        goto  _mkpls0
        clb  SERVO        ; send end of servo pulse
        return

;*****
; Subroutine to get the state of the three input buttons and save in the flags reg.
; Execution time is 38 cycles + (200 us + 6 cycles) per pressed switch (approx.)
;*****

```

```
; The switches are all normally open types and are commoned to one of the pins on port b. The
; other terminal of each switch goes to individual port a pins. This arrangement minimises the
; number of external components required (none). It is important to avoid having more than one
; of the port a button drivers set as an output at any time. Each button is sensed in turn by
; setting its driver line to an output and driving it low. The sense line on port b is
; configured to have a weak pullup and will reflect the complement of the button state. Since
; the lcd is also driven from port b and is unhappy about pullups on it's data and control
; lines (because it already has internal pullups), then the port pullups are only enabled
; during the actual button scan.
```

```
getbuttons  clb    FLG_UP           ; clear the button flags
            clb    FLG_DN
            clb    FLG_MODE
            seb    B_RP0           ; enable pullups on port b so we can sense
            clb    B_NOT_RBPU      ; the switches commoned to the sense pin
            seb    SENSE_DIR       ; set the sense pin to read
            seb    DNS_DIR         ; set downscan pin to input (to avoid contention)
            seb    MODE_DIR        ; and the mode pin
            clb    UPS_DIR         ; set upscan pin to output
            clb    B_RP0
            clb    UPSCAN         ; and pull upscan pin low
            nop
            bbs    SENSE, _gb2     ; is button pressed ? (negative logic)
            seb    FLG_UP         ; yes, then set up flag
            movlf  TEMP2, CLK * 10 ; check switch is stable for 200 us
_gb0       bbc    SENSE, _gb1     ; is button still pressed ?
            clb    FLG_UP         ; no, then reset up flag and stop looking
            goto  _gb2
_gb1       decfsz TEMP2           ; button still pressed, then check again
            goto  _gb0

_gb2       seb    B_RP0           ; now check the downscan pin
            seb    UPS_DIR        ; set the upscan pin to input
            clb    DNS_DIR        ; and set the downscan pin to output
            clb    B_RP0
            clb    DNSCAN        ; and pull the downscan pin low
            nop
            bbs    SENSE, _gb5     ; is button pressed ? (negative logic)
            seb    FLG_DN         ; yes, then set down flag
            movlf  TEMP2, CLK * 10 ; check switch is stable for 200 us
_gb3       bbc    SENSE, _gb4     ; is button still pressed ?
            clb    FLG_DN         ; no, then reset up flag and stop looking
            goto  _gb5
_gb4       decfsz TEMP2           ; button still pressed, then check again
            goto  _gb3

_gb5       seb    B_RP0           ; now check the mode pin
            seb    DNS_DIR        ; set the downscan pin to input
            clb    MODE_DIR       ; and set the mode pin to output
            clb    B_RP0
            clb    MODE           ; and pull the mode pin low
            nop
            bbs    SENSE, _gb8     ; is button pressed ? (negative logic)
            seb    FLG_MODE       ; yes, then set down flag
            movlf  TEMP2, CLK * 10 ; check switch is stable for 200 us
_gb6       bbc    SENSE, _gb7     ; is button still pressed ?
            clb    FLG_MODE       ; no, then reset up flag and stop looking
            goto  _gb8
_gb7       decfsz TEMP2           ; button still pressed, then check again
            goto  _gb6

_gb8       seb    B_RP0           ; switches have now been sensed
            seb    B_NOT_RBPU     ; disable port b pullups
            seb    MODE_DIR       ; leave all pins as inputs
            clb    B_RP0

            return

;*****
; Subroutine to read neutral pulse width from EEPROM and return it in W          *
; The neutral pulse width is the only data stored in the eeprom and is at 0    *
; This is standard manufacturer's recommended code.                          *
;*****
```

```
rdnpw      clb    B_RP0           ; bank 0
            clrf   EEADR          ; address 0
```

```

        seb     B_RP0           ; bank 1
        seb     B_RD           ; EE read
        clb     B_RP0           ; bank 0
        movfw   EEDATA         ; W = data read
        return

;*****
; Subroutine to write neutral pulse width from W to EEPROM
; Routine does not return until the write to EEPROM is complete.
; This is standard manufacturer's recommended code.
;*****

wrnpw   clb     B_RP0           ; bank 0
        clrff  EEADR           ; address 0
        movwf  EEDATA         ; data to write = W
        seb     B_RP0           ; bank 1
        seb     B_WREN         ; set write enable
        movlf  EECON2, 0x55    ; write 55h
        movlf  EECON2, 0xAA    ; write AAh
        seb     B_WR           ; set write bit
        clb     B_WREN         ; inhibit write enable
_wrnpw0 skbc     B_WR           ; loop until write complete
        goto   _wrnpw0
        clb     B_RP0           ; bank 0
        return

;*****
; Subroutine to get the pulse edge times as 24 bit numbers
; If any edge is not detected in approx. 100ms then return with carry flag set
;*****

; Wait for the pulse positive edge from the input sense line representing t1
; The edge is captured as a 24 bit value in T1LO, T1MI, and T1HI

_gpet   movlf   DEL0, TIMEOUT_L ; initialize timeout counter
        movlf   DEL1, TIMEOUT_H
_gpet4  bbs     PULSE, _gpet5    ; loop until pulse line is high or timeout
        incf   DEL0             ; increment timeout counter low byte
        bbs     PULSE, _gpet5    ; minimise latency
        clrwdt ; clear watchdog timer
        bbs     PULSE, _gpet5    ; minimise latency
        bbc     B_Z, _gpet4      ; did counter low byte overflow ?
        bbs     PULSE, _gpet5    ; yes, minimise latency
        incf   DEL1             ; and increment timeout counter high byte
        bbs     PULSE, _gpet5    ; minimise latency
        bbc     B_Z, _gpet4      ; did counter high byte overflow ?
        seb     B_C             ; yes, then set the carry flag to indicate
        return ; timeout and return
_gpet5  movff   TMR0, T1LO       ; capture timer low byte
        movff   TIMERM, T1MI     ; save the remainder of the 24 bit time
        movff   TIMERH, T1HI
        nop
        movf   PLS_CTR           ; do not send first x pulses to servo
        bz     _gpet6           ; if counter zero then send all pulses
        decf   PLS_CTR         ; else decrement the counter
        goto  $+2              ; and skip
_gpet6  seb     SERVO           ; send positive edge to servo

; Wait for the pulse negative edge from the input sense line representing t2
; The edge is captured as a 24 bit value in T2LO, T2MI, and T2HI

_gpet7  movlf   DEL0, TIMEOUT_L ; initialize timeout counter
        movlf   DEL1, TIMEOUT_H
        bbc     PULSE, _gpet8    ; loop until pulse line is low or timeout
        incf   DEL0             ; increment timeout counter low byte
        bbc     PULSE, _gpet8    ; minimise latency
        clrwdt ; clear watchdog timer
        bbc     PULSE, _gpet8    ; minimise latency
        bbc     B_Z, _gpet7      ; did counter low byte overflow ?
        bbc     PULSE, _gpet8    ; yes, minimise latency
        incf   DEL1             ; and increment timeout counter high byte
        bbc     PULSE, _gpet8    ; minimise latency
        bbc     B_Z, _gpet7      ; did counter high byte overflow ?
        seb     B_C             ; yes, then set the carry flag to indicate
        return ; timeout and return
_gpet8  movff   TMR0, T2LO       ; capture timer low byte

```

```

        movlf    TMR0, 2                ; reset and adjust for 2 cycle timer latency
        movff   TIMERM, T2MI           ; save the remainder of the 24 bit time
        movff   TIMERH, T2HI
        movf    PLS_CTR                ; do not send first x pulses to servo
        bz     _gpet9                  ; if counter zero then send all pulses
        decf   PLS_CTR                 ; else decrement the counter
        goto   $+2                      ; and skip
_gpet9  clb    SERVO                    ; send negative edge to servo
        clrf   TIMERM                  ; reset timer middle and high bytes
        clrf   TIMERH
        clb    B_C                      ; clear carry to signal success
        return                          ; and return

;*****
; Subroutine to initialise the lcd for 4 bit operation
; Procedure is "Initialisation by instruction" applicable to Phillips PCF2116X
;*****

lcd_init  clrf    LCDPORT                ; set data directions for lcd port
          seb    B_RP0
          movlf  LCDTRIS, LCDWRITE
          clb    B_RP0

          ; Initialization by instruction
          delay 100                      ; wait > 15ms after Vdd rises above Vpor
          movlf  LCDPORT, SET_8B >> 4
          tgl_clk                ; set the mode to 8 bit data
          delay 20                  ; wait more than 4.1ms
          movlf  LCDPORT, SET_8B >> 4
          tgl_clk                ; set the mode to 8 bit data
          delay 5                    ; wait more than 100us
          movlf  LCDPORT, SET_8B >> 4
          tgl_clk                ; set the mode to 8 bit data
          delay 5                    ; wait more than 100us
          movlf  LCDPORT, SET_4B_2L >> 4
          tgl_clk                ; set the mode to 4 bit data and 2 lines
          movlf  LCDPORT, SET_4B_2L & 0x0F
          tgl_clk
          delay 5
          movlw  DISP_ON              ; busy status valid from here on
          call  lcd_cmd                ; turn on display and no cursor or blink
          clear_disp                  ; clear the display
          movlw  ENTRY_INC             ; set entry mode to increment/no shift
          call  lcd_cmd
          movlw  DD_RAM_ADDR           ; set ddram address to 0
          call  lcd_cmd
          return

;*****
; Subroutine to send a command to the lcd (4 bit bus)
; Enter with the byte in W
; Uses TEMP1 (not preserved)
;*****

lcd_cmd   movwf   TEMP1                ; save the command
          call   lcd_busy              ; wait until command can be accepted
          swapf  TEMP1, W              ; upper nibble to lower
          andlw  0x0F                  ; mask off upper nibble to zero
          movwf  LCDPORT               ; put the byte on the port
          tgl_clk                ; toggle lcd data bus clock
          movfw  TEMP1                 ; retrieve the original byte
          andlw  0x0F                  ; mask upper nibble
          movwf  LCDPORT               ; put the byte on the port
          tgl_clk                ; toggle lcd data bus clock
          return

;*****
; Subroutine to send a character to the lcd (4 bit bus)
; Enter with the character in W
; Uses TEMP1 (not preserved)
;*****

lcd_char  movwf   TEMP1                ; save the character
          call   lcd_busy              ; wait until character can be accepted
          swapf  TEMP1, W              ; upper nibble to lower
          andlw  0x0F                  ; mask off upper nibble to zero
          movwf  LCDPORT               ; put the byte on the port
          seb    LCD_RS                 ; and set the register select bit

```

```

    tgl_clk          ; toggle lcd data bus clock
    movfw    TEMP1   ; retrieve the original byte
    andlw    0x0F    ; mask upper nibble
    movwf    LCDPORT ; put the byte on the port
    seb     LCD_RS   ; and set the register select bit
    tgl_clk          ; toggle lcd data bus clock
    return

;*****
; Subroutine to send a hexadecimal digit to the lcd (4 bit bus)          *
;   Enter with the digit in W                                           *
;   Uses TEMP1 (not preserved)                                          *
;*****

lcd_hex    andlw    0x0F          ; mask high nibble
           movwf    TEMP1        ; and save result
           movlw    10           ; compare number to 10
           subwf    TEMP1, W     ;
           movlw    0x27        ; if number is > 9 add 0x27
           skbc     B_C          ; to make 'a' to 'f'
           addwf    TEMP1
           addlf    TEMP1, 0x30   ; convert to ascii by adding 0x30
           call    lcd_busy      ; wait until character can be accepted
           swapf    TEMP1, W     ; upper nibble to lower
           andlw    0x0F        ; mask off upper nibble to zero
           movwf    LCDPORT      ; put the byte on the port
           seb     LCD_RS       ; and set the register select bit
           tgl_clk          ; toggle lcd data bus clock
           movfw    TEMP1        ; retrieve the original byte
           andlw    0x0F        ; mask upper nibble
           movwf    LCDPORT      ; put the byte on the port
           seb     LCD_RS       ; and set the register select bit
           tgl_clk          ; toggle lcd data bus clock
           return

;*****
; Subroutine to send a string to the lcd                                 *
;   Enter with the string table offset - 1 in W (ie. first string offset = 0) *
;   The code in the table assumes the offset is in TOFFS                *
;*****

lcd_str    movwf    TOFFS        ; save the offset
_lcdstr0   call    str_table
           andlw    0xFF        ; check for end of string
           skbc     B_Z        ; zero returned at end
           return              ; yes, done
           call    lcd_char     ; else display the char
           incf    TOFFS        ; point to next char
           goto    _lcdstr0

;*****
; Subroutine to set the DDRAM address to a character position in a line *
;   Enter with the address in W                                         *
;   The scheme is as follows;                                           *
;   W <0:5>    address in line - can be 0 to 0x27 (40 chars, 16 visible) *
;   W <6>      0 = line 1, 1 = line 2                                   *
;   Uses TEMP1 and W (not preserved)                                    *
;*****

lcd_sladdr movwf    TEMP1        ; save address
           btfss   TEMP1, 6      ; check if line 2
           goto    _sladdr0     ; line 1, do no more
           bcf    TEMP1, 6      ; line 2, clear flag and add 0x40
           addlf  TEMP1, 0x40
_sladdr0   bsf    TEMP1, 7      ; set the high bit
           call    lcd_busy      ; wait until address can be accepted
           swapf    TEMP1, W     ; upper nibble to lower
           andlw    0x0F        ; mask off upper nibble to zero
           movwf    LCDPORT      ; put the byte on the port
           tgl_clk          ; toggle lcd data bus clock
           movfw    TEMP1        ; retrieve the original byte
           andlw    0x0F        ; mask upper nibble
           movwf    LCDPORT      ; put the byte on the port
           tgl_clk          ; toggle lcd data bus clock
           return

;*****

```

```

; Subroutine to read the busy flag and wait until lcd not busy
;*****
lcd_busy    clrfs    LCDPORT                ; set data directions for a read
            sebs    B_RP0
            movlfs LCDTRIS, LCDREAD
            clb     B_RP0

_busy0     clrwdt                ; clear watchdog timer
            clb     LCD_RS         ; set lcd for command mode
            sebs    LCD_RW         ; setup to read busy flag
            tgl_clk                ; toggle lcd data bus clock
            bbc     LCD_DB7, _busy1 ; check busy bit
            tgl_clk                ; toggle lcd data bus clock
            goto    _busy0         ; loop to poll the busy bit again
_busy1     tgl_clk                ; busy bit is clear so send rest of cmd
            clrfs    LCDPORT        ; reset data directions
            sebs    B_RP0
            movlfs LCDTRIS, LCDWRITE
            clb     B_RP0
            return

;*****
; Subroutines to shift the lcd display one character
;*****

lcd_shr     movlws SHR_C            ; shift display one character to right
            call    lcd_cmd
            return

lcd_shl     movlws SHL_C            ; shift display one character to left
            call    lcd_cmd
            return

;*****
; Subroutine to shift the lcd display 16 characters to the left
;*****

lcd_shl16   movlfs TEMP2, 0x10      ; shift display left 16 to reveal new page
_shl16     call    lcd_shl
            delay   50              ; delay for 50ms, 800ms for entire shift
            decfsz TEMP2            ; loop if not done
            goto    _shl16
            return

;*****
; Binary To BCD Conversion Routine
; This routine converts a 16 Bit binary Number to a 5 Digit BCD Number.
;
; The 16 bit binary number is input in locations HIBYTE and LOBYTE with the high
; byte in HIBYTE.
; The 5 digit BCD number is returned in BCD0, BCD1 and BCD2 with BCD0 containing
; the MSD in its right most nibble.
; Routine uses TEMP1 and TEMP2 (not preserved)
;*
; Performance : Program Memory :      35
;               Clock Cycles   :     885
;*****

b2_bcd     clrwdt
            clb     B_C            ; clear the carry bit
            movlfs TEMP2, 16
            clrfs    BCD0
            clrfs    BCD1
            clrfs    BCD2

_loop16    rlf     LOBYTE
            rlf     HIBYTE
            rlf     BCD2
            rlf     BCD1
            rlf     BCD0
            decfsz TEMP2
            goto    _adjDEC
            retlw   0

_adjDEC    movlfs FSR, BCD2
            call    _adjBCD
            movlfs FSR, BCD1
            call    _adjBCD
            movlfs FSR, BCD0
            call    _adjBCD

```

```

_adjBCD    goto    _loop16
           movlw   3
           addwf  INDF, W
           movwf  TEMP1
           btfsc  TEMP1, 3           ; test if result > 7
           movwf  INDF
           movlw  0x30
           addwf  INDF, W
           movwf  TEMP1
           btfsc  TEMP1, 7           ; test if result > 7
           movwf  INDF               ; save as MSD
           retlw  0

;*****
; Subroutine to delay for precise 10 us x value in W register (incl. call & return)      *
; minus 10 cycles for values of W from 3 to 255 inclusive.                            *
; Total cycles = 10W x CLK / 4 - 10 = 10W us - 10 cycles                             *
; total bytes = 5 x CLK / 4 + 11 = 16MHz/31bytes, 8MHz/21bytes, 4MHz/16bytes          *
;*****

           ;call   udelay             ; 2 (placeholder for call)
udelay    movwf  DEL0                 ; 1
           decf   DEL0                 ; 1
           decf   DEL0                 ; 1

_udeLAY0  variable count = 5 * CLK / 4 - 2
           while  count > 0
           goto  $+1                   ; 2 * (W - 2) x (5 x CLK / 4 - 2)
           count--
           endw
           nop                          ; W
           decfsz DEL0                  ; W - 2 + 1
           goto  _udeLAY0              ; 2 * (W - 2) - 2

           movlw  5 * CLK / 4 - 4       ; 1
           movwf  DEL0                 ; 1
_udeLAY1  nop                          ; 5 * CLK / 4 - 4
           decfsz DEL0                  ; 5 * CLK / 4 - 4 + 1
           goto  _udeLAY1              ; 2 * (5 * CLK / 4 - 4) - 2
           clrwdt                       ; 1
           return                       ; 2

;*****
; Subroutine to delay for precise 1 ms x value in W register (incl. call & return)      *
; for values of W from 2 to 255 inclusive.                                           *
; Total cycles = 1,000W x CLK / 4 - 4 = W ms - 4 cycles                             *
; total bytes = 12                                                                *
;*****

           ;call   sdelay             ; 2 (placeholder for call)
sdelay    movwf  DEL1                 ; 1
           decf   DEL1                 ; 1
_udeLAY0  goto  $+1                   ; 2W - 2
           goto  $+1                   ; 2W - 2
           goto  $+1                   ; 2W - 2
           movlw  100                   ; W - 1
           call   udelay                ; (W - 1) x (1,000 x CLK / 4 - 10)
           decfsz DEL1                  ; W - 1 + 1
           goto  _udeLAY0              ; 2W - 2 - 2
           movlw  100                   ; 1
           call   udelay                ; 1,000 x CLK / 4 - 10
           return                       ; 2

;*****
; Subroutine to delay for precise 100 ms x value in W register (incl.call & return)    *
; for values of W from 2 to 255 inclusive.                                           *
; Total cycles = 100,000W x CLK / 4 + 2 = 100W ms + 2 cycles                       *
; total bytes = 9                                                                *
;*****

           ;call   ldelay             ; 2 (placeholder for call)
ldelay    movwf  DEL2                 ; 1
           decf   DEL2                 ; 1
_udeLAY0  movlw  100                   ; W - 1
           call   sdelay                ; (W - 1) x (100,000 x CLK / 4 - 4)
           decfsz DEL2                  ; W - 1 + 1
           goto  _ldelay0              ; 2W - 2 - 2

```

```

        movlw    100                ; 1
        call    sdelay              ; 100,000 x CLK / 4 - 4
        return   2                  ; 2

;*****
; Strings for display (note: limit of 256 bytes)
; Entry with the offset to the string from str0 in TOFFS
; The whole of str_table and the strings must be within a single page of memory
; Here we use the top page of memory
; On call, the program counter high byte will be loaded from pclath on execution
; of the first instruction which modifies the pc. Prior to this, pclath must be
; loaded with the high byte of the page address of the table.
;*****

        org     endpgm - 164        ; put strings in top page of memory

str_table  movlw    $ >> 8          ; get the high byte of the table address
          movwf   PCLATH            ; put it in program counter latch high byte
          movfw   TOFFS            ; retrieve the offset
          addwf   PCL               ; and add to pc - pclath is loaded here

str0      ;!.....!.....!..... Display positions
str1      dt      "R/C Pulse Meter from Eagle Air", 0
str2      dt      " Version ", SOFT_VER + 0x30, ".", SOFT_REV + 0x30, " "
          dt      " Australia P/L", 0
str3      dt      "Pulse = 1.500 ms", 0
str4      dt      "Frame = 20.00 ms", 0
str5      dt      "No Input Signal", 0
str6      dt      "Output: 1.500 ms", 0
str7      dt      "Neutral Pulse", 0
str8      dt      "Set to: 1.500 ms", 0

        end

```