

```

;*****
; Program      : MIXERSMT.ASM
; Function     : R/C Servo Mixer (SMT Version)
; Author      : Engr. John F. Fitter B.E.
; Language    : PIC Assembler
; Platform    : PIC 16C84, 16F84 or 16C61
; Revisions   : 28apr97 Original
;             : 24may97 Fixed problem with sensing ch1 pulse leading edge
;             : 01jun98 Revised for new pcb layout
;
; Copyright c 1997-98 Eagle Air Australia Pty. Ltd. All rights reserved
; (Some code fragments are based on application notes and other public sources)
;*****

; This program mixes two servo signals to produce sum and difference signals. The signals can
; be mixed with each other or into each other as described below. Input signals are pulse
; trains repeated at 20 to 25ms intervals. Each pulse is nominally 1.5ms +/- 0.5ms.

; Pulses are represented in the program by signed integers representing the pulse width
; deviation from the neutral pulse width in processor clock cycles (fosc/4).

; If r is the mixing ratio, a number from 0 to 1, and ChN is the signed deviation of channel N
; from the neutral pulse width, then;

; Mixing Y with X, a proportion of ChY is bi-directionally mixed with ChX
;
;           ChA = (ChX + r * ChY) / (1 + r)
;           ChB = (ChX - r * ChY) / (1 + r)

; Mixing Y into X, a proportion of ChY is uni-directionally mixed into ChX
;
;           ChA = (ChX + r * ChY) / (1 + r)
;           ChB = ChY

; If R is an integer from 1 to 8 representing the proportion of mix with 8 being a mixing ratio
; of 1, then the relations are as follows;

; Mixing Y with X           ChA = (8 * ChX + R * ChY) / (8 + R)
;                           ChB = (8 * ChX - R * ChY) / (8 + R)
; Mixing Y into X           ChA = (8 * ChX + R * ChY) / (8 + R)
;                           ChB = ChY

; This is the form used by the program.

; An example of mixing "with" is flaperons where aileron input causes the flaperons to move in
; opposite directions and flap input causes the flaperons to move in the same direction.
; An example of mixing "into" is maneuvering flaps where elevator input causes the flaps to
; move but flap input effects only the flaps and not the elevator.

; A DIL6 configuration switch is used for field alteration of operating parameters.
; Each switch is wired as normally open / close to ground.

; Switch      On              Off

; 1,2,3      Mix ratio (3 bit)
; 4          Mix with              Mix into
; 5          Mix Ch1 with/into Ch0  Mix Ch0 with/into Ch1
; 6          ChB normal            ChB reversed

; Mix ratio is specified in eighths with zero being 8/8ths mix. To get zero mix
; do not install the mixer !!

; Stimulus files:   mixer01.sti   Input A = 1.5ms/20ms   Input B = 1.5ms/20ms

;*****
; Program equates
;*****

list          p=16F84, r=dec, b=6, x=OFF
expand

; System types are assembler command line defines.
; futaba, jr, multiplex, hitech, psa

CPU          equ          1684
SIM          equ          0           ; Change timing consts for simulator
CLK         equ          4000        ; clock speed in KHz
include     "e16f84.inc"           ; include enhanced defines for 16C84
__config    _CP_OFF & _PWRTE_ON & _WDT_ON & _HS_OSC ; configuration fuses

```

```

        ifdef          futaba
PW_CTR      equ        1200          ; Centre pulse width in us
PW_DEVN     equ        500          ; Pulse width deviation in us
        else
        ifdef          multiplex
PW_CTR      equ        1350
PW_DEVN     equ        500
        else
PW_CTR      equ        1500          ; default: JR, PSA, Airtronics
PW_DEVN     equ        500
        endif
        endif
        ; multiplex
        ; futaba

CPW         equ        PW_CTR * CLK / 4000 ; Centre pulse width in clock cycles
PWD         equ        PW_DEVN * CLK / 4000 ; Pulse width deviation in cycles

FALSE      equ        0
TRUE       equ        1
        ; Logical equates

INI_INTCON equ        B'00100000'      ; Interrupt control register
        ; initial values
        ; T0IE = 1 (TMR0 o/f int enabled)
        ; others = 0

INI_OPTION equ        B'10001010'      ; Option register initial values
        ; *RBPU = 1 (PortB pullups disabled)
        ; PSA = 1 (prescaler to WDT)
        ; WDT = 2 (WDT rate = 1:4)
        ; others = 0

;*****
; Port pin assignments
;*****

INI_DIRA    equ        0
INI_DIRB    equ        0xff
        ; Initial PA dir's - all outputs
        ; Initial PB dir's - all inputs

        bit          B_CHA,    3,      PORTA    ; Channel A Output port bit
        bit          B_CHB,    2,      PORTA    ; Channel B Output port bit
        bit          B_CH0,    4,      PORTB    ; Channel 0 Input port bit
        bit          B_CH1,    3,      PORTB    ; Channel 1 Input port bit
        bit          B_RAT0,    0,      PORTB    ; Mix ratio lsb
        bit          B_RAT1,    1,      PORTB    ; Mix ratio isb
        bit          B_RAT2,    2,      PORTB    ; Mix ratio msb
        bit          B_MXTYP,   5,      PORTB    ; Mixing type
        bit          B_MXORD,   6,      PORTB    ; Mixing order
        bit          B_BDIR,    7,      PORTB    ; ChB direction

;*****
; Working registers
;*****

user_regs   org        0x0c
HIBYTE      res        1
LOBYTE      res        1
EXBYTE      res        1
TEMP        res        3
TEMP0       res        2
TEMP1       res        2
BITCOUNT   res        1
CH0_IN      res        2
CH1_IN      res        2
CHA_OUT     res        2
CHB_OUT     res        2
MIX_RATIO   res        1
TIMER_H     res        1
FLAGS       res        1
        bit          B_SIGN,    0,      FLAGS    ; Sign flag for signed arithmetic
        bit          B_MIXTYP,   1,      FLAGS    ; Mixing type flag
        bit          B_MIXORD,   2,      FLAGS    ; Mixing order flag
        bit          B_BREV,     3,      FLAGS    ; B reverse flag

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

; Macro to move 16 bit literal to register pair _dest16 and _dest16+1

```

```

movlf16    macro           _dest16, _literal16
            movlw          HIGH((_literal16)&0xffff)
            movwf         (_dest16)&0x7f
            movlw          LOW((_literal16)&0xffff)
            movwf         (_dest16+1)&0x7f
            endm

; Macro to move 16 bit source register pair to _dest16 and _dest16+1

movff16    macro           _src16, _dest16
            movf           (_src16)&0x7f, W
            movwf         (_dest16)&0x7f
            movf           (_src16+1)&0x7f, W
            movwf         (_dest16+1)&0x7f
            endm

;*****
; Interrupt service routine - only TM0 interrupts are used
;*****

intvec     org             _intvec
intvec     clb             B_TOIF           ; clear the interrupt flag
intvec     incf            TIMER_H, F      ; maintain the 16 bit timer
intvec     retfie

;*****
; Processor initialisation
;*****

init       seb            B_RP0           ; select bank 1
init       movlf          OPTION_REG, INI_OPTION ; option bits
init       movlf          TRISA, INI_DIRA  ; data directions for port A
init       movlf          TRISB, INI_DIRB  ; data directions for port B
init       clb            B_RP0           ; select bank 0
init       clb            INTCON          ; clear interrupt control register
init       seb            B_TOIE          ; and enable timer interrupts
init       clb            PORTA           ; clear port A latches
init       clb            PORTB           ; clear port B latches

;*****
; Main program loop
;*****

main
main_init  clrwdt          ; clear watchdog (72ms timeout)
main_init  call           pulse_in        ; get the input pulse widths
main_init  call           get_config      ; get the configuration switches
main_init  call           calc_pw         ; calculate input pulse deviations
main_init  cbs            B_MIXORD, swap_inputs ; swap channels if order is 0 i/w 1
main_init  call           mix_ch          ; perform the mixing
main_init  cbs            B_BREV, reverse_B ; reverse ChB if required
main_init  call           pulse_out       ; send the output pulses
main_init  goto           main_init       ; end of main program loop

;*****
; Name: pulse_in
; Procedure to acquire the input channels 0 and 1 pulse widths.
;
; The procedure is designed for speed and accuracy and so is written in inline code. The
; procedure polls the input channel bit until it goes high then zeroes the timer and waits
; for the input bit to go low at which time it reads the timer. This method has fewer
; overheads than using Port B change interrupts and uses about the same amount of program
; code. Furthermore, Port B change interrupts have a timing uncertainty of 3 cycles for
; internal synchronisation which is no better than the polled method. Other problems with the
; interrupt driven method are related to the following;
;
; Reading a 16 bit timer
; The pic16c84 only has an 8 bit timer. To maintain a 16 bit timer the program must manage
; the upper 8 bits in software and increment the high byte in the timer overflow interrupt
; service routine. This works well. Reading the timer is the problem because two bytes must
; be read while the timer continues to run. If a read process is begun just before timer
; overflow then the high byte will be incremented in the middle of the process and the result
; will be in error by 256. Even if the interrupts are disabled the same problem will occur
; because of timer rollover without high byte updating.
;
; The solution is to stop the timer which can be done for three instruction cycles by writing

```

```

; to the timer register. OR'ing the timer with zero accomplishes this and the counter is      *
; stopped long enough to read the 16 bit value.                                          *
;                                                                                          *
;*****
pulse_in    clrf      TIMER_H          ; clear timer high byte
            clrwdt
            bbs      B_CH0, $-2        ; ensure Ch0 input is low
            clrwdt
            bbc      B_CH0, $-2        ; wait for Ch0 to go high
            clrf     TMR0              ; clear the timer low byte,
            clb      B_TOIF           ; and the overflow flag
            seb      B_GIE            ; and enable the timer interrupts
            clrwdt
            bbs      B_CH0, $-2        ; then wait for Ch0 to go low
            clrw
            iorwf    TMR0, F          ; write to TMR0 without changing it
            movf     TMR0, W          ; then read TMR0 - the clock is now
            movwf    CH0_IN+1        ; stopped for 3 instruction cycles
            movf     TIMER_H, W      ; so quickly grab the timer count
            movwf    CH0_IN
            clb      B_GIE            ; then disable the interrupts
            bbs      B_GIE, $-2        ; (recommended method for pic16c84)

            clrf     TIMER_H          ; clear timer high byte
            clrwdt
            bbc      B_CH1, $-2        ; wait for Ch1 to go high
            clrf     TMR0              ; clear the timer low byte,
            clb      B_TOIF           ; and the overflow flag
            seb      B_GIE            ; and enable the timer interrupts
            clrwdt
            bbs      B_CH1, $-2        ; then wait for Ch1 to go low
            clrw
            iorwf    TMR0, F          ; write to TMR0 without changing it
            movf     TMR0, W          ; then read TMR0 - the clock is now
            movwf    CH1_IN+1        ; stopped for 3 instruction cycles
            movf     TIMER_H, W      ; so quickly grab the timer count
            movwf    CH1_IN
            clb      B_GIE            ; then disable the interrupts
            bbs      B_GIE, $-2        ; (recommended method for pic16c84)

            return

;*****
; Name: mix_ch                                     *
; Procedure to mix channels 0 and 1 to produce channels A and B                       *
;                                                                                          *
;   Registers used = W, LOBYTE, HIGHBYTE, EXBYTE, TEMP, TEMP+1, TEMP+2, TEMP0, TEMP1   *
;   Stack usage    = 2                            *
;*****
mix_ch      movff    MIX_RATIO, TEMP    ; compute R * ChY
            movff16  CH1_IN, HIBYTE
            call     mpy16b8
            movff16  HIBYTE, TEMP1      ; and save in a safe place
            movlf    TEMP, 8            ; compute 8 * ChX
            movff16  CH0_IN, HIBYTE
            call     mpy16b8
            movff16  HIBYTE, TEMP0      ; and save in another safe place
            movff16  TEMP1, TEMP        ; compute 8 * ChX + R * ChY
            call     add16
            movff    MIX_RATIO, TEMP
            addlf    TEMP, 8            ; compute 8 + R
            call     div16b8            ; ChA
            movff16  HIBYTE, CHA_OUT
            bbc      B_MIXTYP, _mix_ch0 ; if mixing type is "mix with"
            movff16  CH1_IN, CHB_OUT   ; then ChB = ChY
            return

_mix_ch0   movff16  TEMP0, HIBYTE       ; retrieve 8 * ChX
            movff16  TEMP1, TEMP        ; retrieve R * ChY
            call     sub16              ; compute 8 * ChX - R * ChY
            movff    MIX_RATIO, TEMP
            addlf    TEMP, 8            ; compute 8 + R
            call     div16b8            ; ChB
            movff16  HIBYTE, CHB_OUT
            return

```

```

;*****
; Name: pulse_out
; Procedure to send the Channels A and B output pulses
;
; Registers used = W, LOBYTE, HIGHBYTE, TEMP, TEMP+1
; Stack useage = ???
;*****

pulse_out    movff16    CHA_OUT, HIBYTE        ; ChA
             movlf16    TEMP, CPW
             call       add16                    ; add in the centre pulse width
             seb        B_CHA                    ; set output high
             call       delay                    ; leave high for HIBYTE:LOBYTE cycles
             clb        B_CHA
             movff16    CHB_OUT, HIBYTE        ; ChB
             movlf16    TEMP, CPW
             call       add16                    ; add in the centre pulse width
             seb        B_CHB                    ; set output high
             call       delay                    ; leave high for HIBYTE:LOBYTE cycles
             clb        B_CHB
             return

;*****
; Name: delay
; Procedure to delay HIBYTE:LOBYTE cycles
;
; The procedure enables the timer and timer overflow interrupts. The timer is operated as a
; 16 bit up counter and is preloaded with the negative of the desired timeout value in cycles
; adjusted for overheads.
; Overheads are ; 2 cycles to call procedure
;                31 cycles to load the timer and start it
;                2 cycles to respond to timer high byte overflow
;                7 cycles to tidy up and return (if GIE resets first try)
;                1 cycle in calling procedure to reset the output bit
;*****

delay        movlf16    TEMP, -43                ; 4 allow for overheads
             call       add16                    ; 9
             call       neg16                    ; 10 negate time to allow up count
             movff      HIBYTE, TIMER_H          ; 2 load the timer registers
             clrf       TMR0                    ; 1 prevent premature interrupts
             clb        B_T0IF                  ; 1 clear any pending overflows
             seb        B_GIE                    ; 1 enable interrupts
             movff      LOBYTE, TMR0            ; 2 timing starts from here
;             seb        B_Z                        ; 1 force loops to 1st interrupt
;_delay0     bnz        _delay0                  ; 2 loop until TIMER_H overflows
_delay0      movf       TIMER_H, W
             bnz        _delay0
             clb        B_GIE                    ; 1 disable the interrupts
             bbs        B_GIE, $-2              ; 4 (recommended method for pic16c84)
             return                               ; 2

;*****
; Name: get_config
; Procedure to get the configuration switches and save them in the flags register and the
; mixing ratio register
;
; A significant feature here is the changing of the sense port data directions. In order to
; sense the switches the pins must be inputs and the pullups turned on, however because the
; pullups must be turned off for normal operation of the mixer and some or all of the
; switches may be open, noise on these pins would cause unnecessary port B change interrupts
; and lead to servo jitter. The solution is to make the switch sense pins outputs when not
; being sensed and drive them low (safe because the switches only switch to ground).
;
; Registers used = W, MIX_RATIO, FLAGS
; Stack useage = nil
;*****

get_config   seb        B_RP0                    ; set data directions to all inputs
             movlf      TRISB, 0xff              ; then turn ON the Port B pullups
             clb        B_NOT_RBPU              ; because of the pullups, switches
             clb        B_RP0                    ; are logic high when open - logic
             clrf       FLAGS                    ; is converted to positive logic here
             skbs      B_MXTYP
             seb        B_MIXTYP                ; get the configuration switches into
             skbs      B_MXORD                    ; the flags register - slow method
             seb        B_MIXORD                ; but wiring independent

```

```

    skbs      B_BDIR
    seb      B_BREV
    clrf     MIX_RATIO
    skbs      B_RATIO          ; get the mixing ratio - slow method
    bsf      MIX_RATIO, 0      ; but wiring independent
    skbs      B_RATIO1        ; the ratio is in 1/8ths from 1 to 8
    bsf      MIX_RATIO, 1
    skbs      B_RATIO2
    bsf      MIX_RATIO, 2
    seb      B_RP0            ; turn OFF Port B pullups - do not
    seb      B_NOT_RBPU       ; leave pullups on because port B is
    movlf    TRISB, INI_DIRB  ; connected to the receiver output
    clb      B_RP0            ; then restore data directions
    movlw    8
    movf     MIX_RATIO, F      ; if mixing ratio is zero
    skpznz
    movwf    MIX_RATIO
    return

;*****
; Name: calc_pw
; Procedure to calculate the pulse width deviations of channels 0 and 1
;
; Registers used = W, LOBYTE, HIGHBYTE, TEMP, TEMP+1
; Stack usage = 1
;*****

calc_pw    movff16    CH0_IN, HIBYTE          ; Ch0 -= centre pulse width
           movlf16    TEMP, -CPW
           call       add16
           movff16    HIBYTE, CH0_IN
           movff16    CH1_IN, HIBYTE        ; Ch1 -= centre pulse width
           call       add16
           movff16    HIBYTE, CH1_IN
           return

;*****
; Name: reverse_B
; Procedure to reverse the direction of output Channel B
;
; Registers used = W, LOBYTE, HIGHBYTE
; Stack usage = 1
;*****

reverse_B  movff16    CHB_OUT, HIBYTE
           call       neg16
           movff16    HIBYTE, CHB_OUT
           return

;*****
; Name: swap_inputs
; Procedure to swap the input registers for Ch0 and Ch1
;
; Registers used = W, CH0_IN, CH1_IN, TEMP, TEMP+1
; Stack usage = nil
;*****

swap_inputs movff16    CH0_IN, TEMP
           movff16    CH1_IN, CH0_IN
           movff16    TEMP, CH1_IN
           return

;*****
; Name: sub16, add16 (2 different entry points)
; Procedures to add and subtract two 16 bit numbers
;
; Entry:      operand 1 in HIBYTE:LOBYTE
;             operand 2 in TEMP:TEMP+1
; Exit:       HIBYTE:LOBYTE += TEMP(0:1) if entry at add16
;             HIBYTE:LOBYTE -= TEMP(0:1) if entry at sub16
;             carry status is same as 8 bit add or subtract
;
; Registers used = W, TEMP, TEMP+1, HIBYTE, LOBYTE
; Total cycles = 15 for sub16 and 9 for add16
; Total bytes = 12 for sub16 and 7 for add16
; Stack usage = nil
;*****

```

```

sub16      comf      TEMP+1, F          ; negate TEMP(0:1)
           incf      TEMP+1, F
           skpnz
           decf      TEMP, F
           comf      TEMP, F
add16      movf      TEMP+1, W          ; add TEMP(0:1) and HIBYTE:LOBYTE
           addwf     LOBYTE, F          ; add lsb
           skpnc
           incf      HIBYTE, F          ; add in carry
           movf      TEMP, W
           addwf     HIBYTE, F          ; add msb
           return

;*****
; Name: mpy16b8
; Procedure to multiply a signed 16 bit and an unsigned 8 bit number to produce
; a 24 bit signed product.
;
; Entry:      multiplicand in HIBYTE:LOBYTE
;            multiplier in TEMP
; Exit:       product in EXBYTE:HIBYTE:LOBYTE
;
; Registers used = W, TEMP, TEMP+1, TEMP+2, HIBYTE, LOBYTE, EXBYTE, BITCOUNT
; Total cycles = 188 (worst case, 0xffff * 0xff)
; Total bytes = 34
; Stack usage = 1
;*****

mpy16b8    clb       B_SIGN              ; save the sign of the result
           btfsc    HIBYTE, 7
           seb      B_SIGN
           skbs     B_SIGN              ; is HL +ve ?
           goto     _mpy16b80           ; yes, then do the multiply
           call     neg16               ; no, then make it +ve
_mpy16b80  movlwf   BITCOUNT, 16        ; initialize the bit counter
           movff16  HIBYTE, TEMP+1      ; save temporary multiplicand
           clrf     HIBYTE              ; clear multiplicand
           clrf     LOBYTE
           movf     TEMP, W              ; save the multiplier in W
_mpy16b81  rrf      TEMP+1, F            ; shift temporary multiplicand to the
           rrf      TEMP+2, F            ; right into carry
           skpnc
           addwf    EXBYTE, F            ; if carry set then need to add
           rrf      EXBYTE, F            ; add C:EXBYTE += TEMP
           rrf      HIBYTE, F            ; shift multiplicand and carry right
           rrf      LOBYTE, F
           decfsz  BITCOUNT, F         ; any more bits ?
           goto     _mpy16b81           ; yes, loop until all bits checked
           skbs     B_SIGN              ; if B_SIGN set then answer is -ve
           return                       ; return on +ve
           comf     LOBYTE, F            ; else, negate the product
           comf     HIBYTE, F
           comf     EXBYTE, F
           incf     LOBYTE, F
           skpnz
           incf     HIBYTE, F
           skpnz
           incf     EXBYTE, F
           return

;*****
; Name: div16b8
; Procedure to divide a signed 16 bit number by an unsigned 8 bit number to
; produce a 16 bit signed quotient and an 8 bit unsigned remainder.
;
; Entry:      numerator in HIBYTE:LOBYTE
;            denominator in TEMP
; Exit:       quotient in HIBYTE:LOBYTE
;            remainder in EXBYTE
;
; It is important that before calling this procedure to ensure that the numerator is
; greater than the denominator. If this is not so then either the numerator or the
; denominator should be pre-scaled to ensure this restriction is met.
;
; Registers used = W, TEMP, TEMP+1, TEMP+2, HIBYTE, LOBYTE, EXBYTE, BITCOUNT
; Total cycles = 274 (worst case)
;*****

```

```

; Total bytes      = 32
; Stack usage     = 1
;*****

div16b8      clb          B_SIGN          ; save the sign of the result
             btfsc       HIBYTE, 7
             seb         B_SIGN
             skbs        B_SIGN          ; is HL +ve ?
             goto        _div16b80      ; yes, then do the divide
             call        neg16          ; no, then make it +ve
_div16b80    movlf       BITCOUNT, 16  ; for 8 shifts
             movff16    HIBYTE, TEMP+1 ; save the temporary numerator
             clrf        HIBYTE         ; clear the numerator
             clrf        LOBYTE
             clrf        EXBYTE         ; clear the remainder
_div16b81    clb         B_C
             rlf         TEMP+2, F      ; rotate numerator to left
             rlf         TEMP+1, F
             rlf         EXBYTE, F      ; rotate carry into partial remainder
             movf        TEMP, W
             subwf       EXBYTE, W      ; is denominator > partial remainder?
             skpc        ; carry set if yes
             goto        _div16b82      ; else don't shift in a bit
             movf        TEMP, W        ; partial remainder -= denominator
             seb         B_C            ; shift a bit into the quotient
_div16b82    rlf         LOBYTE, F
             rlf         HIBYTE, F
             decfsz     BITCOUNT, F    ; loop untill all bits checked
             goto        _div16b81
             skbs        B_SIGN        ; if B_SIGN set then answer is -ve
             return      ; return on +ve
             call        neg16          ; else, negate the quotient
             return

;*****
; Name: neg16
; Procedure to negate a 16 bit signed number
;
; Entry:      16 bit number to be negated in HIBYTE:LOBYTE
; Exit:       result in HIBYTE:LOBYTE
;
; Registers used = HIBYTE, LOBYTE
; Total cycles = 8
; Total bytes = 6
; Stack usage = nil
;*****

neg16        comf        LOBYTE, F      ; negate HIBYTE:LOBYTE
             comf        HIBYTE, F
             incf        LOBYTE, F
             skpnz
             incf        HIBYTE, F
             return

end
; end of program

```