



## **MicroMixer**

Microprocessor controlled mixer for  
Radio Control Systems  
*(Surface mount technology, Version 2.0)*

### **Description**

The MicroMixer is an electronic device whose function is to mix two servo signals from a conventional pulse width modulation (PWM or PPM) radio control receiver to produce the sum and difference signals.

The MicroMixer performs all of its functions within a small microprocessor under software control. Other than the microprocessor there are only two other components, a ceramic resonator to provide a clock reference for the microprocessor and a six way dual inline switch to enable the user to set various operating parameters for the software.

The MicroMixer is encased in heatshrink plastic tubing with a cutout for the dual inline switch access. Two signal input leads enter at one end and two signal output leads emerge from the other end.

The MicroMixer is extremely functional for such a simple circuit and is one of the few, if not the only mixer to implement a mathematically correct mixing algorithm.

### **Basic Theory**

Conventional recreational radio control systems use a system of data transmission known as Pulse Position Modulation (PPM), also known as Pulse Width Modulation (PWM). This term describes the manner in which the control position information is packaged prior to modulation onto the RF carrier and transmission to the target model.

In R/C terminology a channel refers to a single control, so a transmitter capable of sending the positions of 8 controls is called an 8 channel transmitter. The control positions are sent as serial pulses, one pulse for each control and packaged into a nominal 20mS frame. Each pulse is nominally 1.5mS wide and full control deflection will vary the pulse width from 1mS to 2mS.

The simplest receiver decoder simply routes the pulses to the appropriate servos. Servo is R/C terminology for the control positioner, which accepts the pulses from the receiver decoder and positions its mechanical output according to the pulse width. A

single servo therefore sees a series of pulses, each arriving every 20mS and of a duration of 1 to 2mS.

## Mixing

The purpose of a mixer is to combine two channels of data from a radio control receiver in a linear fashion to produce two channels of information for delivery to the servos. This mixer can do this in one of two different ways.

Mixing "into" involves taking a fraction of one channel and adding it to the other channel. An example of "mixing into" would be manoeuvring flaps where an elevator input causes flaps to move but a flap input has no effect on the elevator.

Mixing "with" involves adding a fraction of one channel into the other and subtracting a fraction of the other channel from the first. An example of "mixing with" would be flaperons where an aileron input causes the flaperons to move differentially and a flap input causes the flaperons to move together. Other examples are V-tails (rudder and elevator) and Delta (elevator and aileron). A more esoteric use is to mix aileron into elevator on an aerobatic airplane to prevent torsional fatigue of the airframe.

If we call the two input channels ChX and ChY and the two outputs ChA and ChB and then consider for the moment the most basic form of mixing.

ChX and ChY are added and the result divided by two to produce the first output, ChA. Similarly, ChX and ChY are subtracted and the result which is the difference between ChX and ChY is divided by two to produce the second output, ChB. This scheme works well and it is not possible to drive the servos past their limits. It is however, limited to mixing equal amounts of ChX and ChY.

In most applications it is desirable to be able to alter the amount of mixing to a figure other than 100% (equal amounts). The problem is then the divide ratio, which is no longer two. The sum and difference signal must be divided by a figure that provides for maximum servo deflection without over-ranging. To do this the following formula is used;

Mixing Y with X, a proportion of ChY is bi-directionally mixed with ChX

$$\text{ChA} = (\text{ChX} + r * \text{ChY}) / (1 + r)$$

$$\text{ChB} = (\text{ChX} - r * \text{ChY}) / (1 + r)$$

Mixing Y into X, a proportion of ChY is uni-directionally mixed into ChX

$$\text{ChA} = (\text{ChX} + r * \text{ChY}) / (1 + r)$$

$$\text{ChB} = \text{ChY}$$

where r is the mixing ratio, a number from 0 to 1

The program in the MicroMixer implements these formulas for mixing ratios from one to eight eighths. A mixing ratio of zero is not catered for since not using the mixer would accomplish the same thing.

## Operation

### Setting the switches

The MicroMixer has six switches that must be set prior to use. Each switch is either ON or OFF and the relevant position is shown printed on the switch body. The function of each of the switches is as follows;

Switch	On	Off
1,2,3	Mix ratio (3 bit)	
4	Mix with	Mix into
5	Mix ChY with/into ChX	Mix ChX with/into ChY
6	ChB normal	ChB reversed

The mix ratio is specified in eighths with zero being 8/8ths mix. The switches are set as a binary representation in the following manner;

Ratio	Sw1	Sw2	Sw3
8/8	Off	Off	Off
1/8	On	Off	Off
2/8	Off	On	Off
3/8	On	On	Off
4/8	Off	Off	On
5/8	On	Off	On
6/8	Off	On	On
7/8	On	On	On

### Connecting the mixer

The mixer is supplied with two input leads at one end and two output leads at the other end. Connect the two servos to the output leads. Plug the two input leads into the receiver sockets that correspond to the channels to be mixed.

Power the radio system and ensure that the mixer is functioning correctly. If so, then the installation is complete.

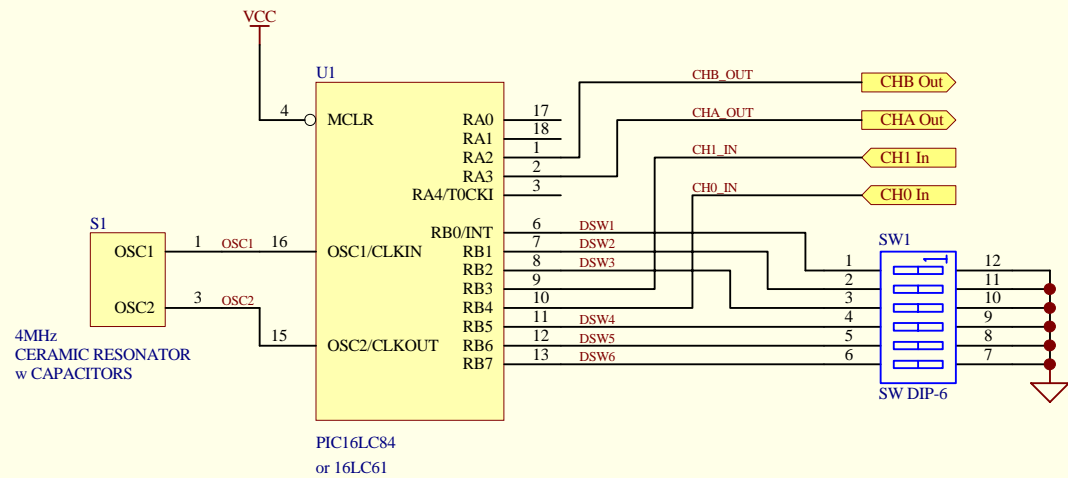
On rare occasions the servos will operate slowly and with a jerking motion. This is a consequence of how the software in the microprocessor measures the width of the input pulses. If the two input channels are very close to each other in the pulse sequence there may not be enough time for the processor to complete it's calculations before the next pulse arrives. Under these conditions it takes two frames to acquire the two input pulses with the result that the output pulses are produced at half the frequency that they should be and the servo response is therefore sluggish.

The cure is simple. Reverse the input connections and reset the switches to suit the new arrangement.

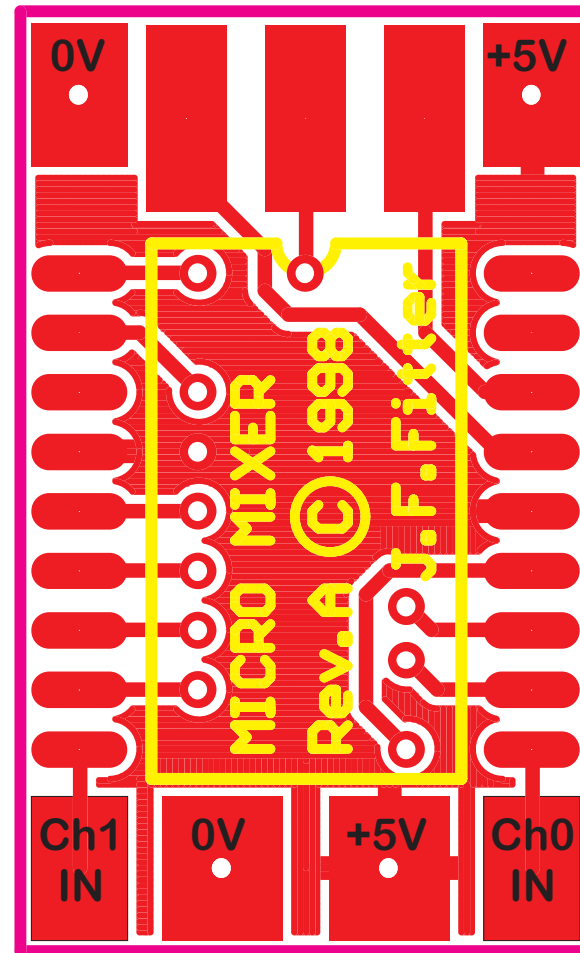
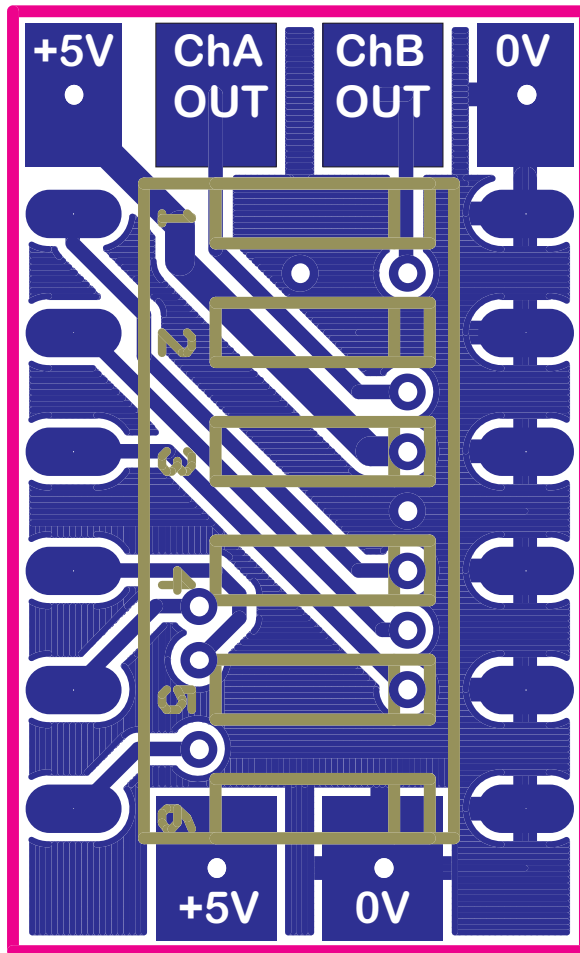
## Specifications

The following basic specifications are applicable to the MicroMixer V1.0 currently shipping.

Input/Output	2 channels of pulse width modulated information in and out 1500us neutral pulse width, 500us deviation, 20ms frame width. Constructors with a programmer can alter this by simply changing the equates in the program. Persons purchasing prebuilt units or precoded micros can specify variations.
User interface	6 way dual inline switch.
Mixing ratios	1 to 8 in eighths, selectable by switch
Mix types	Mix into and mix with, selectable by switch
Mixing order	Selectable by switch
Reversing	One output channel can be reversed, selectable by switch
Resolution	1us equivalent to 10 bits over the servo operating range or 11 bits total
Power	2.5 to 6.5volts at 2mA nominal
Size	12mm x 17mm
Construction	Fibreglass pcb, double sided & plated through holes, silkscreened and solder masked, surface mount technology.
Connectors	Solder pads



Title			
RC MIXER - Schematic			
Size	Number	Revision	
A4	EA0007003010698A1	A1	
Date:	4-Mar-1999	Sheet of	1
File:	E:\commercl\mixer\mixer.sch	Drawn By:	John F. Fitter B.E.



View from underside of PCB

View from top of PCB

General information	do not scale/dimensions in mm tolerances 0.05 angles 0.1 general draft unspecified radii geometric tolerances to BS308 remove all burr and sharp edges	Material	0.8mm FRP			Title	RC MIXER - Composite and Connections				
	EAGLE AIR AUSTRALIA PTY. LTD.	Finish	Epoxy			Size	A4		Number	EA0007001010698A1	Revision
						Date:	1-jun-1998		Sheet	1 of 1	
						File:	mixerpcb.cdr		Drawn By:	JFF	

A

B

C

D

4

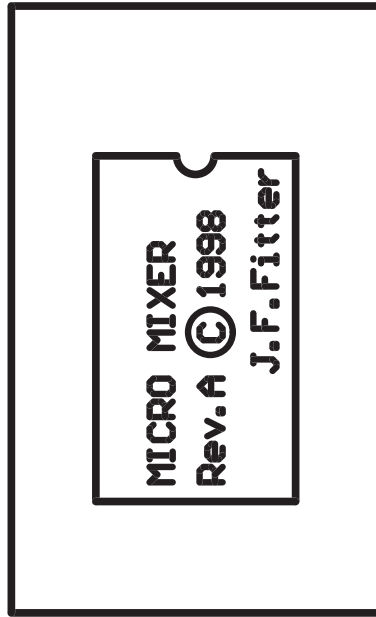
4

3

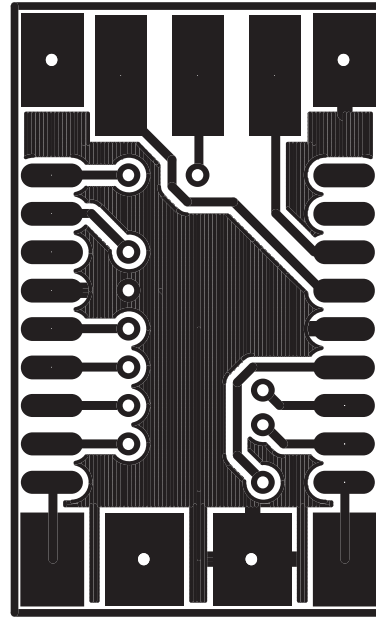
3

2

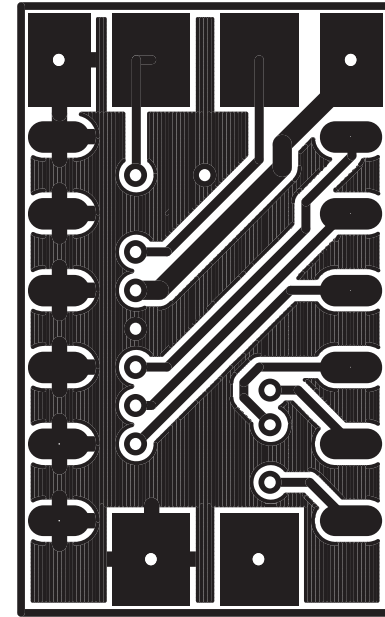
2



Top o er



Top opper



Bottom opper

1

1

General information	do not scale/dimensions in mm tolerances 0.05 angles 0.1 general draft unspecified radii geometric tolerances to BS308 remove all burr and sharp edges	Material	0.8mm FRP		Title		RC MIXER - Final art	
		Finish	Epoxy		Size	Number	Revision	
EAGLE AIR AUSTRALIA PTY. LTD.					A4	EA0007002010698A1	A1	
					Date:	1-jun-1998	Sheet	1 of 1
					File:	mixerfin.cdr	Drawn By:	JFF

A

B

C

D

```

;*****
; 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
        endif          ; 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                ; Initial PA dir's - all outputs
INI_DIRB    equ        0xff            ; 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                ; arithmetic working registers
LOBYTE      res        1
EXBYTE      res        1
TEMP        res        3
TEMP0       res        2
TEMP1       res        2
BITCOUNT   res        1
CH0_IN      res        2                ; Channel 0 Input register 16b
CH1_IN      res        2                ; Channel 1 Input register 16b
CHA_OUT     res        2                ; Channel A Output register 16b
CHB_OUT     res        2                ; Channel B Output register 16b
MIX_RATIO   res        1                ; Proportion of mixing
TIMER_H     res        1                ; Timer high byte
FLAGS       res        1                ; Program bit flags
        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 usage = ???
;*****

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 usage = 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
           movfff16 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

```

```

list
; E16F84.INC Enhanced Header File, Version 1.00 Eagle Air Australia P/L
nolist

;*****
; E16C84.INC Enhanced Header File, Version 1.00 *
; This header file provides enhanced defines for PIC 16C84 assembler *
; It replaces p16f84.inc *
; *
; Author : John F. Fitter B.E. *
; Revisions : 17jul96 Original *
; 23jul96 Consolidated p16f84 defines and added new macros *
; *
; Copyright c 1996 Eagle Air Australia Pty. Ltd. All rights reserved *
; (some code fragments are based on application notes and other public sources) *
;*****

        ifndef    _E16F84
        #define   _E16F84

        ifndef    __16F84
        MESSG     "Processor-header file mismatch. Verify selected processor."
        endif

true     equ      1
false   equ      0

input   equ      1
output  equ      0

_resvec equ      0x00      ; 16f84 start address
_intvec equ      0x04      ; 16f84 peripheral interrupt vector
_endpgm equ      0x3FF     ; end of program memory

resetvector org    _resvec      ; reset vector
resvec      goto   init

endmemory  org    _endpgm      ; end of memory
endpgm

;*****
; Macros to implement new addressing modes *
;*****

w        equ      0          ; Working register destination
W        equ      w
f        equ      1          ; File register destination
F        equ      f

movff    macro      src, dest      ; Move register file src to register
        movf        src, W          ; file dest through w
        movwf       (dest)&0x7f
        endm

movlf    macro      dest, k        ; Move literal to register file
        movlw       k
        movwf       (dest)&0x7f
        endm

andlf    macro      dest, k        ; AND literal with register file
        movlw       k
        andwf       (dest)&0x7f, f
        endm

iorlf    macro      dest, k        ; Inclusive OR literal with register file
        movlw       k
        iorwf       (dest)&0x7f, f
        endm

xorlf    macro      dest, k        ; Exclusive OR literal with register file
        movlw       k
        xorwf       (dest)&0x7f, f
        endm

addlf    macro      dest, k        ; ADD literal to register file
        movlw       k
        addwf       (dest)&0x7f, f
        endm

sublf    macro      dest, k        ; SUBTRACT literal from register file
        movlw       k
        subwf       (dest)&0x7f, f
        endm

```

```

;*****
; Macros to set/clear/branch/skip on bits
; These macros define and use synthetic "bit labels"
; Bit labels contain the address and bit of a location
;*****
;
; Usage Description
; -----
; bit label, bit, file ; Define a bit label
; seb label ; set bit using bit label
; clb label ; clear bit using bit label
; skbs label ; SKIP on bit set
; skbc label ; SKIP on bit clear
; bbs label, address ; BRANCH on bit set
; bbc label, address ; BRANCH on bit clear
; cbs label, address ; CALL on bit set
; cbc label, address ; CALL on bit clear

bit macro label, bit, file ; Define a bit label
label equ ((file)&0x7f)<<8|bit; (macro)
endm

seb macro label ; Set bit
bsf label>>8, label&7 ; (macro)
endm

clb macro label ; Clear bit
bcf label>>8, label&7 ; (macro)
endm

skbs macro label ; Skip on bit set
btfss label>>8, label&7 ; (macro)
endm

skbc macro label ; Skip on bit clear
btfsc label>>8, label&7 ; (macro)
endm

bbs macro label, address ; Branch on bit set
btfsc label>>8, label&7 ; (macro)
goto address ; (macro)
endm

bbc macro label, address ; Branch on bit clear
btfss label>>8, label&7 ; (macro)
goto address ; (macro)
endm

cbs macro label, address ; Call on bit set
btfsc label>>8, label&7 ; (macro)
call address ; (macro)
endm

cbc macro label, address ; Call on bit clear
btfss label>>8, label&7 ; (macro)
call address ; (macro)
endm

;*****
; REGISTER FILE DECLARATIONS
;*****

INDF equ 0x00 ; Uses contents of FSR to address data memory
TMRO equ 0x01 ; Timer 0
PCL equ 0x02 ; Program counter low order 8 bits
STATUS equ 0x03 ; Status register
FSR equ 0x04 ; Indirect data memory address pointer
PORTA equ 0x05 ; Port A
PORTB equ 0x06 ; Port B
EEDATA equ 0x08 ; EEPROM data register
EEADR equ 0x09 ; EEPROM address register
PCLATH equ 0x0A ; Write buffer for upper 5 bits of PC
INTCON equ 0x0B ; Interrupt control register

OPTION_REG equ 0x81 ; Option register
TRISA equ 0x85 ; Port A data direction register
TRISB equ 0x86 ; Port B data direction register
EECON1 equ 0x88 ; EEPROM control register 1
EECON2 equ 0x89 ; EEPROM control register 2

CONFIG_REG equ 0x2007 ; Configuration word (no access by program)

;*****
; REGISTER BIT DECLARATIONS
;*****

```

```

bit    B_C,      0, STATUS      ; Carry
bit    B_DC,     1, STATUS      ; Half carry
bit    B_Z,      2, STATUS      ; Zero
bit    B_PD,     3, STATUS      ; Power down
bit    B_TO,     4, STATUS      ; Timeout
bit    B_RP0,    5, STATUS      ; Register bank select bit 0 (direct)
bit    B_RP1,    6, STATUS      ; Register bank select bit 1 (direct)
bit    B_IRP,    7, STATUS      ; Register bank select bit (indirect)

bit    B_RBIF,   0, INTCON      ; RB port change interrupt flag
bit    B_INTF,   1, INTCON      ; RB0/INT interrupt flag
bit    B_TOIF,   2, INTCON      ; TMR0 overflow interrupt flag
bit    B_RBIE,   3, INTCON      ; RB port change interrupt enable
bit    B_INTE,   4, INTCON      ; RB0/INT interrupt enable
bit    B_TOIE,   5, INTCON      ; TMR0 overflow interrupt enable
bit    B_EEIE,   6, INTCON      ; EE write complete interrupt enable
bit    B_GIE,    7, INTCON      ; Global interrupt enable

bit    B_PS0,    0, OPTION_REG  ; prescaler bit 0
bit    B_PS1,    1, OPTION_REG  ; prescaler bit 1
bit    B_PS2,    2, OPTION_REG  ; prescaler bit 2
bit    B_PSA,    3, OPTION_REG  ; prescaler assignment
bit    B_T0SE,   4, OPTION_REG  ; TMR0 source edge select
bit    B_T0CS,   5, OPTION_REG  ; TMR0 clock source select
bit    B_INTEDG, 6, OPTION_REG  ; Interrupt edge select
bit    B_NOT_RBPU, 7, OPTION_REG ; Port B pullup enable

bit    B_RD,     0, EECON1      ; EEPROM Read control bit
bit    B_WR,     1, EECON1      ; EEPROM Write control bit
bit    B_WREN,   2, EECON1      ; EEPROM Write enable bit
bit    B_WRERR,  3, EECON1      ; EEPROM Error flag bit
bit    B_EEIF,   4, EECON1      ; EEPROM Write operation interrupt flag

;*****
; RAM Definition
;*****

    __MAXRAM    H'CF'
    __BADRAM    H'07', H'50'-H'7F', H'87'

;*****
; Configuration Bits
;*****

_CP_ON      EQU    H'000F'
_CP_OFF     EQU    H'3FFF'
_PWRTE_ON   EQU    H'3FF7'
_PWRTE_OFF  EQU    H'3FFF'
_WDT_ON     EQU    H'3FFF'
_WDT_OFF    EQU    H'3FFB'
_LP_OSC     EQU    H'3FFC'
_XT_OSC     EQU    H'3FFD'
_HS_OSC     EQU    H'3FFE'
_RC_OSC     EQU    H'3FFF'

endif ; _E16F84
list

```