```
 title  "Voltage-Controlled Tap Tempo LFO V2D"
;----------------------------------------------------------
;       ELECTRIC DRUID TAP TEMPO LFO VERSION 2D
;----------------------------------------------------------
; Copyright Tom Wiltshire for Electric Druid, May/July 2009
; Modifications by Chris Safi January 2010
;
;  This program provides a versatile Tap Tempo LFO on a single chip.
;  Analogue output is provided as a PWM output, which requires LP
;  filtering to be useable.
;
;  Modifications include the addition of the Wave Distort feature in place of
;  the earlier Pulse Width CV which only affected the Pulse Waveform.
;
;  Hardware Notes:
;   PIC16F684 running at 20 MHz using an external clock
;   RA0: 0-5V Tempo CV
;   RA1/AN1: 0-5V Waveform select CV
;   Only the top three bits are used, and the waves are encoded as follows:
;       0 - Ramp Up
;       1 - Ramp Down
;       2 - Pulse
;       3 - Triangle
;       4 - Sine
;       5 - Sweep wave ("logarithmic" sweep)
;       6 - Lumps wave
;       7 - Random S&H
;   RA2/AN2: 0-5V Tap Tempo multiplier CV
;   Only the top three bits are used, and the waves are encoded as follows:
;       0 - x0.5        Half note
;       1 - x1          Quarter note (the default, and what tempo is based on)
;       2 - x1.5        Quarter note triplet
;       3 - x2          1/8th note
;       4 - x3          1/8th note triplet
;       5 - x4          1/16th note
;       6 - x1  unused
;       7 - x1  unused
;   RA3: 0-5V Tap tempo digital frequency input
;   RA4/RA5: Clock Oscillator 20MHz Xtal
;   RC0/AN4: 0-5V Output level control
;   RC1/AN5: 0-5V Wave Distort CV (Adjusts duty cycle of all waveforms)
;   RC2: 0-5V 'Next Multiplier' digital input (steps through multipliers)
;   RC3: Clock Output (an auxillary 0-5V squarewave output at LFO Freq)
;   RC4: Tempo LED (Indicates tap timing - LED is on when timing)
;   RC5: PWM LFO Output
;
; (2B and 2C never amounted to anything)
;
; Version 2D - 8th June 2011
; Modified the way that the Clock output works to prevent it from being able to
; drift out of sync.Improved the noise gen (simpler and faster).
;

 LIST R=DEC
 INCLUDE "p16f684.inc"

 __CONFIG _FCMEN_OFF & _IESO_OFF & _BOD_OFF & _CPD_OFF & _CP_OFF & _MCLRE_OFF & _PWRTE_ON
& _WDT_OFF & _HS_OSC


;----------------------------------------------------------
;       Variables
```

```
;---------------------------------------------------------

 CBLOCK 0x020
        ; Registers for context saving during interrupts
        W_TEMP
        STATUS_TEMP
        PCLATH_TEMP
        FSR_TEMP
        ; General working storage
        TEMP            ; Used for the AD delay
        WAVETEMP        ; The sine lookup needs a temp register
        LOOKUPTEMP      ; The FREQ_INC lookup needs a temp register too
;       NOISETEMP       ; Noise source needs temp storage for XOR output
        ; Various flag bits. 1 is on, obviously.
        FLAGS           ; See Define statements below for details
        ; Input debouncing routine for Tap Tempo input
        DEBOUNCE_HI     ; Debounce vertical counter
        DEBOUNCE_LO
        IN_STATE        ; State of all inputs (flags)
        IN_CHANGES      ; Input changes (flags)
        ; The current A/D channel and value
        ADC_CHANNEL
        ADC_VALUE
        ; 16 bit Galois LFSR noise generator
        RAND_HI
        RAND_LO
        ; The 24 bit phase accumulator
        PHASE_HI
        PHASE_MID
        PHASE_LO
        ; The 24 bit phase accumulator (CLOCK)
        PHASE_HI_CLOCK
        PHASE_MID_CLOCK
        PHASE_LO_CLOCK
        ; The 24 bit raw frequency increment
        FREQ_INC_HI
        FREQ_INC_MID
        FREQ_INC_LO
        ; The actual 24 bit phase distortion increments
        FREQ_INC_A_HI   ; This is the inc for the first half of the wave
        FREQ_INC_A_MID
        FREQ_INC_A_LO
        FREQ_INC_B_HI   ; This is the inc for the 2nd half of the wave
        FREQ_INC_B_MID
        FREQ_INC_B_LO
        ; The basic fractional frequency increment
        RAW_INC_HI
        RAW_INC_LO
        ; The current phase distortion CV
        DISTORT_CV
        ; Late addition - The division routine for Phase Distortion
        NUMBER_HI
        NUMBER_MID
        NUMBER_LO
        REMAIN
        DIVISOR
        DIVTEMP         ; Used as a counter in the Division routine

        INCTEMP_HI      ; Used by UpdateFreqIncs to store the raw
        INCTEMP_MID     ; Freq Inc value when it has been x128
        INCTEMP_LO
        TEMPO_CV        ; The current Tempo CV
        TEMPO_UPPER     ; The tempo control movement limits
        TEMPO_LOWER

        MULTIPLIER      ; FREQ_INC = RAW_INC * MULTIPLIER
        MULT_CV         ; The current Multiplier CV
        OLD_MULT_CV     ; The last index from the Multiplier CV knob
        OLD_MULT_SWITCH ; The last index from the Next Multiplier input

        ; The 3 bit waveform select value
        WAVE
        ; The current output level CV
        LEVEL_CV
        ; The current pulse width CV
;       PWM_CV
        ; The 16 bit output level (after it's been scaled by LEVEL_CV)
        OUTPUT_HI
        OUTPUT_LO
        ; The offset that gets added to the output to ensure it is
        ; bipolar and centred around 128
        OFFSET_HI
        OFFSET_LO

        ; The Millisecond counter for the Tap Tempo feature
        MSECS_HI
        MSECS_LO
        ; The 24 by 16 bit division used for the tap tempo
        ; calculation of raw frequency increment
        DIV_HI          ; Dividend
        DIV_MID         ; Although these are variables, in fact we
        DIV_LO          ; only ever divide a constant by the Msecs value.
        ; Divisor  - MSECS_HI:MSECS_LO
        DIV_COUNT       ; Counter
        REM_HI          ; Remainder
        REM_LO
        ; Output is in DIV_HI:DIV_MID:DIV_LO

        ; The 16 by 8 bit multiplication used to turn the raw 1/96th
        ; frequency increment into the final frequency increment
        MULT_HI         ; The output
        MULT_MID
        MULT_LO
        ; The RAW_INC value is used directly, as is MULTIPLIER.
        ; Only the result goes into MULT_xx
 ENDC

;----------------------------------------------------------
;       DEFINE STATEMENTS
;----------------------------------------------------------

; Useful bit definitions for clarity
#define ZEROBIT                 STATUS,Z        ; Zero Flag
#define CARRY                   STATUS,C        ; Carry
#define BORROW                  STATUS,C        ; Borrow is the same as Carry
; Flag bit definitions
#define SECOND_TAP              FLAGS, 0        ; 0=First Tap, 1=Second Tap
#define SEGMENT_END             FLAGS, 1        ; 1=Phase has overflowed
#define TAP_MODE                FLAGS, 2        ; 0=Tempo CV, 1=Tap
;#define MULT_MODE              FLAGS, 3        ; 0=Mult CV, 1=Next Mult Input
#define TAP_TIMER_ON    FLAGS, 3        ; 0=Off, 1=On
; Bits in the debounce variables
#define TAP_STATE               IN_STATE, 3             ; Current debounced state of Tap
```

```
Input
#define TAP_CHANGED             IN_CHANGES, 3   ; Has TAP_STATE changed?
#define MULT_STATE              IN_STATE, 2             ; Debounced state of Next Mult
Input
#define MULT_CHANGED    IN_CHANGES, 2   ; Has MULT_STATE changed?

; Input/Output bit definitions
#define TAP_IN                  PORTA, 3        ; Tap Tempo Input
#define NEXT_MULT_IN    PORTC, 2        ; Next Multiplier Input
#define TEMPO_LED               PORTC, 4        ; Tap Tempo LED
#define CLOCK_OUT               PORTC, 3        ; Clock Pulse Output

;----------------------------------------------------------------------
; Begin Executable Code Segment
;----------------------------------------------------------------------
        org     0x000           ; processor reset vector
        nop                     ; for ICD use
        goto    Main            ; Go to the main program


        org     0x004           ;Interrupt vector location
InterruptEnter:
        movwf   W_TEMP              ; save W register
        swapf   STATUS, W           ; swap status to be saved into W
        bcf     STATUS, RP0         ; ---- Select Bank 0 -----
        movwf   STATUS_TEMP         ; save STATUS register
        movfw   PCLATH
        movwf   PCLATH_TEMP         ; save PCLATH register
        movfw   FSR
        movwf   FSR_TEMP            ; save FSR register

;---------------------------------------------------------
; Interrupt Service Routine (ISR)
; Timer2 ISR deals with the DDS and PWM output
; It also increments the tap timer.
;---------------------------------------------------------

; PWM Timebase at 19.5KHz
Timer2ISR:
        btfss   PIR1, TMR2IF    ; Check if TMR2 interrupt
        goto    InterruptExit
        bcf             PIR1, TMR2IF    ; Clear TMR2 interrupt flag

; Do we need to increment the Tap Timer?
        btfss   TAP_TIMER_ON
        goto    IncrementPhase
        ; Increment the milliseconds counter
        incf    MSECS_LO, f
        btfsc   ZEROBIT         ; If MSEC_LO has overflowed to zero, we've got a carry
        incf    MSECS_HI, f
        btfss   ZEROBIT         ; Has the mSecs counter overflowed?
        goto    IncrementPhase  ; No, so skip
        ; MSECS counter has overflowed. The user has waited too long
        ; between taps, so reset everything.
        bcf             TAP_TIMER_ON    ; Stop the counter
        bcf             SECOND_TAP      ; Next tap is first tap
        bcf             TEMPO_LED       ; Turn off LED

; Increment the DDS phase accumulator PHASE (24+24 bit addition)
IncrementPhase:
        ; Test the high bit in here and branch to one of two versions:
        ; SectionAincrement or SectionBIncrement
        btfsc   PHASE_HI, 7     ; Are we onto the 2nd half of the wave?
        goto    SectionBIncrement

SectionAIncrement:
        clrc
        movf    FREQ_INC_A_LO, w                ; Add FREQ_INC_LO to PHASE_LO
        addwf   PHASE_LO, f
        movf    FREQ_INC_A_MID, w               ; Add FREQ_INC_MID to PHASE_MID
        skpnc
        incfsz  FREQ_INC_A_MID, w
        addwf   PHASE_MID, f
        movf    FREQ_INC_A_HI, w                ; Add FREQ_INC_HI to PHASE_HI
        skpnc
        incfsz  FREQ_INC_A_HI, w
        addwf   PHASE_HI, f
        goto    IncrementPhaseClock             ; Skip Section B

SectionBIncrement:
        clrc
        movf    FREQ_INC_B_LO, w                ; Add FREQ_INC_LO to PHASE_LO
        addwf   PHASE_LO, f
        movf    FREQ_INC_B_MID, w               ; Add FREQ_INC_MID to PHASE_MID
        skpnc
        incfsz  FREQ_INC_B_MID, w
        addwf   PHASE_MID, f
        movf    FREQ_INC_B_HI, w                ; Add FREQ_INC_HI to PHASE_HI
        skpnc
        incfsz  FREQ_INC_B_HI, w
        addwf   PHASE_HI, f

        ; Has the wavecycle ended?
        btfss   CARRY
        goto    IncrementPhaseClock

WavecycleEnded:
        bsf     SEGMENT_END                     ; Tell the S&H waveform
        bsf     CLOCK_OUT                       ; Start a new clock pulse


; Increment the Clock Phase Accumulator
; This is only done for the first 50% of the period, since that's how long the
; output pulse lasts. After that it's ignored, which prevents glitches if the
; clock were to wrap round before the LFO.
IncrementPhaseClock:
; Do we need to bother?
        btfss   CLOCK_OUT               ; Is the Clock output high? Are we on the first
half?
        goto    SelectWaveform  ; No, so skip

        movfw   FREQ_INC_LO             ; Add FREQ_INC_LO to PHASE_LO
        addwf   PHASE_LO_CLOCK, f

        movfw   FREQ_INC_MID    ; Add FREQ_INC_MID to PHASE_MID
        skpnc
        incfsz  FREQ_INC_MID, w
        addwf   PHASE_MID_CLOCK, f

        movfw   FREQ_INC_HI     ; Add FREQ_INC_HI to PHASE_HI
        skpnc
        incfsz  FREQ_INC_HI, w
        addwf   PHASE_HI_CLOCK, f
        ; Right, so are we on the second half now?
        btfss   PHASE_HI_CLOCK, 7
        goto    SelectWaveform          ; No, so skip

ResetClock:
        bcf             CLOCK_OUT                       ; Reset the clock pulse for the
next wavecycle
        clrf    PHASE_LO_CLOCK
        clrf    PHASE_MID_CLOCK
        clrf    PHASE_HI_CLOCK


; Which waveform table should we use?
SelectWaveform:
        movlw   HIGH WaveformBranch
        movwf   PCLATH
        movf    WAVE, w ; Get current waveform
        addwf   PCL, f  ; Increment program counter with waveform value
WaveformBranch:
        goto    RampUp
        goto    RampDown
        goto    Pulse
        goto    Triangle
        goto    Sine
        goto    Sweep
        goto    Lumps
        goto    SampleAndHold
ReturnWithValue:

; Modify the Output level
;----------------------------------------------------------
; This involves multiplying the DDS LFO source output by
; the LEVEL_CV value - an 8 bit x 8 bit multiplication
; Multiply routine from Microchip App Note 26
; Expects number to be multiplied by LEVEL_CV in W
MultiplyByLevelCV:
        clrf    OUTPUT_HI
        clrf    OUTPUT_LO
        clrc    ; Clear carry? Why didn't I know about this?!
        btfsc   LEVEL_CV,0
        addwf   OUTPUT_HI, f
        rrf             OUTPUT_HI, f
        rrf             OUTPUT_LO, f
        btfsc   LEVEL_CV,1
        addwf   OUTPUT_HI, f
        rrf             OUTPUT_HI, f
        rrf             OUTPUT_LO, f
        btfsc   LEVEL_CV,2
        addwf   OUTPUT_HI, f
        rrf             OUTPUT_HI, f
        rrf             OUTPUT_LO, f
        btfsc   LEVEL_CV,3
        addwf   OUTPUT_HI, f
        rrf             OUTPUT_HI, f
        rrf             OUTPUT_LO, f
        btfsc   LEVEL_CV,4
        addwf   OUTPUT_HI, f
        rrf             OUTPUT_HI, f
        rrf             OUTPUT_LO, f
        btfsc   LEVEL_CV,5
        addwf   OUTPUT_HI, f
        rrf             OUTPUT_HI, f
        rrf             OUTPUT_LO, f
        btfsc   LEVEL_CV,6
        addwf   OUTPUT_HI, f
        rrf             OUTPUT_HI, f
        rrf             OUTPUT_LO, f
        btfsc   LEVEL_CV,7
        addwf   OUTPUT_HI, f
        rrf             OUTPUT_HI, f
        rrf             OUTPUT_LO, f

; Add an offset to ensure the LFO output is centred around 128

        ; Work out the offset to add
        comf    LEVEL_CV, w     ; Offset is (256-LEVEL_CV) / 2
        movwf   OFFSET_HI
        clrf    OFFSET_LO
        bcf             CARRY
        rrf             OFFSET_HI, f
        rrf             OFFSET_LO, f
        ; Add the low byte
        movf    OFFSET_LO, w
        addwf   OUTPUT_LO, f
        btfsc   CARRY
        incf    OUTPUT_HI, f
        ; Add the high byte
        movf    OFFSET_HI, w
        addwf   OUTPUT_HI, f

; Set PWM duty cycle
;----------------------------------------------------------
; This puts the value of PWM_DUTY_CYCLE into the appropriate
; registers.
; Note: we can set the duty cycle in registers
; CCP1CON and CCPR1L because they are double
; buffered and the changes will not take affect
; until the next PWM period starts (TMR2 resets).
PWMOutput:
        ; Put the 2 MSBs of OUTPUT_LO into CCP1CON
;       bcf     CARRY    - unnecessary! I throw the low bits away!
        rlf             OUTPUT_LO, w    ; rotate bit 7 into the carry bit

        bsf             CCP1CON, DC1B1  ; set or clear bit 5 of the CCP1CON register
        btfss   CARRY
        bcf             CCP1CON, DC1B1

;       bcf     CARRY
        rlf     OUTPUT_LO, w    ; rotate bit 6 into the carry bit

        bsf             CCP1CON, DC1B0  ; set or clear bit 4 of the CCP1CON register
        btfss   CARRY
        bcf             CCP1CON, DC1B0

        ; Put the high byte into CCPR1L
        movf    OUTPUT_HI, w
        movwf   CCPR1L

;----------------------------------------------------------
InterruptExit:
        movfw   FSR_TEMP            ; restore FSR register
        movwf   FSR
        movfw   PCLATH_TEMP         ; restore PCLATH register
        movwf   PCLATH
        swapf   STATUS_TEMP, w       ; swap status_temp into W, sets bank to original state
        movwf   STATUS               ; restore STATUS register
        swapf   W_TEMP, f
        swapf   W_TEMP, w            ; restore W register

        retfie

;---------------------------------------------------------
; Analogue to Digital conversion subroutine
; This is used by the main code loop
; Returns the converted value in W and ADC_VALUE
;---------------------------------------------------------

DoADConversion:
        ; Short delay whilst the channel settles
        movlw   D'6'                 ; At 4 MHz, a 22 us delay
        movwf   TEMP                 ; (22us = 2us + 6 * 3us + 1us)
        decfsz  TEMP, f
        goto    $-1

        ; Start the conversion
        bsf     ADCON0, GO
        ; Wait for it to finish
        btfsc   ADCON0, GO             ;  Is it done?
        goto   $ - 1

        ;  Read the ADC Value and store it
        movf    ADRESH, w
        movwf   ADC_VALUE
        return


;--------------------------------------------------------
; Tap Tempo Raw Frequency Increment Calculation
; 24-bit by 16-bit division
; This is used by the main code loop
; Freq Inc = 860370 / mSecs
; Thanks to Nikolai Golovchenko and the PIClist
;--------------------------------------------------------
TempoCalculation:
        ; Set up Dividend (always 9177280=0x8C08C0)(previously 860370= 0xD20D2)
        movlw   0x80
        movwf   DIV_HI
        movlw   0x00
        movwf   DIV_MID
        movlw   0x00
        movwf   DIV_LO
        ; Clear remainder and set up loop counter
        CLRF REM_HI
        CLRF REM_LO
        MOVLW D'24'
        MOVWF DIV_COUNT
LOOPU2416
        RLF DIV_LO, W           ;shift dividend left to move next bit to remainder
        RLF DIV_MID, F           ;
        RLF DIV_HI, F           ;
        RLF REM_LO, F            ;shift carry (next dividend bit) into remainder
        RLF REM_HI, F
        RLF DIV_LO, F           ;finish shifting the dividend and save  carry in DIV_LO.0,
                                ;since remainder can be 17 bit long in some cases
                                ;(e.g. 0x800000/0xFFFF). This bit will also serve
                                ;as the next result bit.
        MOVF MSECS_LO, W          ;subtract divisor from 16-bit remainder
        SUBWF REM_LO, F          ;
        MOVF MSECS_HI, W          ;
        BTFSS STATUS, C         ;
        INCFSZ MSECS_HI, W        ;
        SUBWF REM_HI, F          ;

;here we also need to take into account the 17th bit of remainder, which
;is in DIV_LO.0. If we don't have a borrow after subtracting from lower
;16 bits of remainder, then there is no borrow regardless of 17th bit
;value. But, if we have the borrow, then that will depend on 17th bit
;value. If it is 1, then no final borrow will occur. If it is 0, borrow
;will occur. These values match the borrow flag polarity.

        SKPNC                   ;if no borrow after 16 bit subtraction
         BSF DIV_LO, 0          ;then there is no borrow in result. Overwrite
                                ;DIV_LO.0 with 1 to indicate no
                                ;borrow.
                                ;if borrow did occur, DIV_LO.0 already
                                ;holds the final borrow value (0-borrow,
                                ;1-no borrow)
        BTFSC DIV_LO, 0         ;if no borrow after 17-bit subtraction
         GOTO UOK46LL           ;skip remainder restoration.
        ADDWF REM_HI, F          ;restore higher byte of remainder. (w
                                ;contains the value subtracted from it
                                ;previously)
        MOVF MSECS_LO, W          ;restore lower byte of remainder
        ADDWF REM_LO, F          ;
UOK46LL
        DECFSZ DIV_COUNT, f     ;decrement counter
        GOTO LOOPU2416         ;and repeat the loop if not zero.
        RETURN


;---------------------------------------------------------
;       Tempo Multiplication
; The RAW_INC value from the tempo calculation routine
; above is multiplied by the MULTIPLIER value to give the
; final frequency increment value.
; This is a 16-bit by 8-bit multiply, and we use
; the three bytes of the result.
;---------------------------------------------------------
TempoMultiplication:
        clrf    MULT_HI
        clrf    MULT_MID
        clrf    MULT_LO
        movf    MULTIPLIER, w
        clrc     ; Clear carry? Why didn't I know about this?!
        btfsc   RAW_INC_LO,0
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_LO,1
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_LO,2
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_LO,3
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_LO,4
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_LO,5
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_LO,6
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_LO,7
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_HI,0
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_HI,1
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_HI,2
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_HI,3
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_HI,4
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_HI,5
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_HI,6
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        btfsc   RAW_INC_HI,7
        addwf   MULT_HI, f
        rrf             MULT_HI, f
        rrf             MULT_MID, f
        rrf             MULT_LO, f
        return

;---------------------------------------------------------
;       THE MAIN PROGRAM
; This reads the A/D channels and provides
; values for the DDS
;---------------------------------------------------------
Main:
        clrf    PORTC
        movlw   7               ; Turn off Comparators
        movwf   CMCON0

        ; Set up Timer2/PWM
        clrf    TMR2            ; Using TMR2 as a PWM Generator
        movlw   b'00000100'     ; Enable TMR2
        movwf   T2CON
        clrf    CCPR1L          ; Nothing Moving (Yet)

        ; Clear all peripheral interrupts
        ; EEIF, ADIF, CCP1IF, C2IF, C1IF, OSFIF, TMR2IF, TMR1IF
        clrf    PIR1
        ; Enable Peripheral Interrupts
        ; GIE, PEIE, T0IE, INTE, RAIE, T0IF, INTF, RAIF
        movlw   b'11000000'
        movwf   INTCON

        movlw   b'00001100'     ; Enable single channel PWM
        movwf   CCP1CON

        bsf     STATUS, RP0     ; Bank 1

        ; Peripheral Interrupt Enable Register
        movlw   b'00000010'     ; TMR2 Overflow Int
        movwf   PIE1 ^ 0x80
        ; ADC Input on AN0, AN1, AN2, AN4, and AN5
        movlw   b'00110111'
        movwf   ANSEL ^ 0x80
        movlw   b'00100000'     ; Select ADC Clock as Fosc/32
        movwf   ADCON1 ^ 0x80   ; to ensure TAD of 1.6uS

        movlw   D'255'          ; Setup the Limit for the PWM
        movwf   PR2 ^ 0x80      ; Maximum 19.5 KHz Frequency

        movlw   b'111111'       ; All inputs
        movwf   TRISA ^ 0x80

        movlw   b'000111'       ; RC0:RC2 Inputs, RC3:RC5 Outputs
        movwf   TRISC ^ 0x80

        movlw   b'11010011'             ; Set up Timer0 options
        movwf   OPTION_REG ^ 0x80       ; Int Clk with Prescale, Prescale /16

        bcf     STATUS, RP0     ; Bank 0

        ; Set up initial values of the variables
        movlw   D'2'
        movwf   MULTIPLIER      ; Default to quarter notes
        movwf   OLD_MULT_CV
        movwf   OLD_MULT_SWITCH
        movlw   D'255'
        movwf   LEVEL_CV        ; Default to max output
        movlw   D'128'
        movwf   DISTORT_CV      ; Default to 50% Square Wave
        clrf    WAVE    ; 0-Ramp Up 1-Ramp Down, 2-Pulse, 3-Triangle
                        ; 4-Sine, 5-Sweep, 6-Lumps, 7-S&H
        clrf    FLAGS   ; First Tap to start with.
        ; Set up NOISE shift register
        movlw   0xD5            ; Non-zero seed value
        movwf   RAND_HI
        movlw   0xE7
        movwf   RAND_LO
;       clrf    SHIFT_REG0
;       clrf    SHIFT_REG1
;       clrf    SHIFT_REG2
;       movlw   B'01010100'
;       movwf   SHIFT_REG3      ; Non-zero seed value
        ; Set up LFO phase accumulator and Freq Inc
        clrf    PHASE_LO
        clrf    PHASE_MID
        clrf    PHASE_HI
        movlw   D'3'            ; Freq_Inc=3 default
        movwf   FREQ_INC_HI
        movwf   FREQ_INC_MID
        movwf   FREQ_INC_LO
        movlw   D'3'
        movwf   RAW_INC_HI
        movwf   RAW_INC_LO
        clrf    FREQ_INC_A_HI   ; Clear the two actual freq incs
        clrf    FREQ_INC_A_MID
        clrf    FREQ_INC_A_LO
        clrf    FREQ_INC_B_HI
        clrf    FREQ_INC_B_MID
        clrf    FREQ_INC_B_LO
        ; Set up ADC read channel
        clrf    ADC_CHANNEL     ; Start with FREQ_CV


MainLoop:
;----------------------------------------------------------
;       Test and debounce the digital inputs
;----------------------------------------------------------
; Do Scott Dattalo's vertical counter debounce (www.dattalo.com)
; This could debounce eight inputs, but I'm only using two:
; RA3 TAP TEMPO, and RC2 NEXT MULTIPLIER

        ; Increment the vertical counter
        movf    DEBOUNCE_LO, w
        xorwf   DEBOUNCE_HI, f
        comf    DEBOUNCE_LO, f
        ; See if any changes occured
        movf    PORTA, w        ; Get the RA3 input
        andlw   B'001000'
        movwf   TEMP            ; Store this temporary result
        movf    PORTC, w        ; Get the RC2 input
        andlw   B'000100'
        iorwf   TEMP, w         ; Mix the two inputs together
        xorwf   IN_STATE, w     ; Test for changes
        ; Reset the counter if no change occured
        andwf   DEBOUNCE_LO, f
        andwf   DEBOUNCE_HI, f
        ; If there is a pending change and the count has rolled over
        ; to 0, then the change has been filtered
        xorlw   0xFF            ; Invert the changes
        iorwf   DEBOUNCE_HI, w  ; If count is 0, both..
        iorwf   DEBOUNCE_LO, w  ; ..A and B bits are 0
        ; Any bit in W that is clear at this point means that the input
        ; has changed and the count rolled over
        xorlw   0xFF            ; Invert the changes
        xorwf   IN_STATE, f     ; Update the changes
        ; W is left holding state of all filtered bits that have changed.
        movwf   IN_CHANGES

;---------------------------------------------------------
; Deal with the NEXT MULTIPLIER Input
;---------------------------------------------------------
; This moves us to the next valid multiplier value
NextMultiplierInput:
        ; Has the NEXT MULTIPLIER Input changed state?
        btfss   MULT_CHANGED
        goto    TapTempoInput   ; No, so skip
        ; Has it gone low (pressed)?
        btfsc   MULT_STATE
        goto    TapTempoInput   ; No, so skip

        ; The NEXT MULTIPLIER button has been pressed
        incf    OLD_MULT_SWITCH,f       ; Increment the last index
        ; Limit it to the 0-5 range
        movfw   OLD_MULT_SWITCH ; Get the last index
        xorlw   D'6'            ; Is the index 6?
        btfsc   ZEROBIT         ; If not, skip
        clrf    OLD_MULT_SWITCH ; Index=6, so reset to zero

        ; Now do the call with the multiplier index (0-5)
        movfw   OLD_MULT_SWITCH
        call    GetMultiplier
        movwf   MULTIPLIER      ; Store the value
        ; Multiply the basic RAW_INC Freq Inc by this value
        call    TempoMultiplication
        ; Store the result as the new FREQ_INC
        movfw   MULT_HI
        movwf   FREQ_INC_HI
        movfw   MULT_MID
        movwf   FREQ_INC_MID
        movfw   MULT_LO
        movwf   FREQ_INC_LO
        ; Apply wave distortion to this frequency increment
        goto    UpdateFreqIncs

;---------------------------------------------------------
; Deal with the TAP TEMPO Input
;---------------------------------------------------------
; I'm only interested in when the TAP_IN goes low (button pressed)
; and not in when it is released, so that is ignored.
TapTempoInput:
        ; Has the TAP TEMPO input changed state?
        btfss   TAP_CHANGED
        goto    NextADChannel   ; No, so skip
        ; Has it gone low (pressed)?
        btfsc   TAP_STATE
        goto    NextADChannel   ; No, so skip

; TAP TEMPO has been pressed
```

```
        ; Which tap was it?
        btfsc   SECOND_TAP      ; Is this the second tap?
        goto    SecondTap

; First tap - start the timer to measure between the two taps
FirstTap:
        ; Clear milliseconds counter
        clrf    MSECS_HI
        clrf    MSECS_LO
        ; Start the tap timer
        bsf     TAP_TIMER_ON
        bsf     TEMPO_LED       ; Turn on an LED
        bsf     SECOND_TAP      ; Next tap is the second tap
        ; Reset LFO to beginning of cycle
        clrf    PHASE_HI
        clrf    PHASE_MID
        clrf    PHASE_LO
        ; Reset Clock to beginning of cycle
        clrf    PHASE_HI_CLOCK
        clrf    PHASE_MID_CLOCK
        clrf    PHASE_LO_CLOCK
        ; A phase reset is the same as accumulator overflow,
        ; but only if we haven't just done one
        bsf     SEGMENT_END
        goto    NextADChannel

; Second tap - stop the timer, and use the measured time to work out a new freq inc
SecondTap:
        ; Stop the tap timer
        bcf     TAP_TIMER_ON
        bcf     TEMPO_LED       ; Turn off the LED
        bcf     SECOND_TAP      ; Next tap is the first tap
        ; Do a calculation of 9177280/mSecs to find required RAW_INC
        call    TempoCalculation
        movf    DIV_MID, w
        movwf   RAW_INC_HI
        movf    DIV_LO, w
        movwf   RAW_INC_LO
        ; Multiply the RAW_INC by the Tempo MULTIPLIER to get
        ; the final frequency increment
        call    TempoMultiplication
        ; Store the result as the new FREQ_INC
        movf    MULT_HI, w
        movwf   FREQ_INC_HI
        movf    MULT_MID, w
        movwf   FREQ_INC_MID
        movf    MULT_LO, w
        movwf   FREQ_INC_LO
        ; Two valid taps means we're in Tap mode
        bsf     TAP_MODE        ; Set Tap Tempo mode
        ; Set up limits for the Tempo CV
        ; Set Upper limit
        ; hi = tempo cv + 4
        ; addlw 255-Hi
        movlw   D'4'
        addwf   TEMPO_CV, w
        movwf   TEMPO_UPPER
        comf    TEMPO_UPPER, f  ; Turn Hi into 255-Hi

        ; Set lower limit
        ; lo = tempo_cv -4
        ; addlw (Hi-lo)+1
        ; Note that this always gives me 9, since hi-lo cancels
        ; the tempo cv.
        movlw   D'9'
        movwf   TEMPO_LOWER

        goto UpdateFreqIncs



;---------------------------------------------------------
;       Deal with CV inputs
;---------------------------------------------------------

NextADChannel:
        ; Change to next A/D channel
        incf    ADC_CHANNEL, f

; We need to do different things depending on which value we're reading:
; Tempo         - Use value as a lookup to get a RAW_INC value
; Waveform      - Store top 3 bits as WAVE
; Multiplier    - Use top 3 bits to lookup tempo multiplier
; Output level  - Store it as LEVEL_CV for the multiply routine
; Wave Distort  - Store it as DISTORT_CV for the wave distort phase accumulator adjustment

SelectADCChannel:
        movlw   HIGH SelectADCBranch
        movwf   PCLATH
        movf    ADC_CHANNEL, w  ; Get current channel
        andlw   D'7'            ; Only need three LSBs
        addwf   PCL, f          ; Increment program counter with channel value
SelectADCBranch:
        goto    TempoCV
        goto    WaveformSelectCV
        goto    MultiplierCV
        goto    OutputLevelCV
        goto    PhaseDistortionCV
        movlw   D'7'            ; When we get to 5, we'll jump to here
        movwf   ADC_CHANNEL
        goto    MainLoop

; Update the Tempo
TempoCV:
        movlw   b'00000001'     ; AN0, ADC On
        movwf   ADCON0
        call    DoADConversion
        movwf   TEMPO_CV        ; DEBUG- Store this for now
        ; Which mode are we in? 0=Tempo CV mode, 1=Tap mode
        btfss   TAP_MODE
        goto    TempoCVChanged  ; Tempo CV mode, so go directly

        ; We're in Tap mode, so has the user moved the control?
        addwf   TEMPO_UPPER, w
        addwf   TEMPO_LOWER, w
        ; Carry is clear if tempo CV is outside those limits
        btfsc   CARRY
        goto    MainLoop        ; Carry set, so Tempo CV has not moved
        bcf     TAP_MODE        ; Switch to Tempo CV mode

; The Tempo CV has been altered
TempoCVChanged:
        ; Use the ADC result to get the RAW_INC value
        call    GetFreqIncHi    ; Uses index direct from ADC_VALUE
        movwf   RAW_INC_HI
        call    GetFreqIncLo
        movwf   RAW_INC_LO
        ; Multiply the basic Freq Inc by this value
        call    TempoMultiplication     ; RAW_INC * MULTIPLIER = FREQ_INC
        ; Store the result as the new FREQ_INC
        movfw   MULT_HI
        movwf   FREQ_INC_HI
        movfw   MULT_MID
        movwf   FREQ_INC_MID
        movfw   MULT_LO
        movwf   FREQ_INC_LO
        goto    UpdateFreqIncs


; Update the Waveform
WaveformSelectCV:
        movlw   b'00000101'     ; AN1, ADC On
        movwf   ADCON0
        call    DoADConversion
        ; Apply hysteresis to the CV by checking if ADC_VALUE and ADC_VALUE+4 give the
same result
        addlw   D'4'            ; Add 4 to the AD value in W
        andlw   B'11100000'     ; Get the top 3 bits (Result 1)
        movwf   TEMP
        movf    ADC_VALUE, w
        andlw   B'11100000'     ; Get the top 3 bits (Result 2)
        xorwf   TEMP, w         ; Test if the two results are the same
        btfss   ZEROBIT         ; If zero, they're the same
        goto    MainLoop        ; Not the same, no need to change waveform

        ; We need to change the waveform
        swapf   ADC_VALUE, f    ; We only need the top 3 bits
        rrf     ADC_VALUE, f
        movlw   B'00000111'     ; Mask unwanted bits
        andwf   ADC_VALUE, w
        movwf   WAVE
        goto    MainLoop

; Update the Multiplier CV
MultiplierCV:
        movlw   b'00001001'     ; AN2, ADC On
        movwf   ADCON0
        call    DoADConversion

; DEBUG- Disable this routine
;       goto    MainLoop
;
;       movwf   MULT_CV         ; Store the raw CV value
        ; Apply hysteresis to the CV by checking if ADC_VALUE and ADC_VALUE+4 give the
same result
        addlw   D'4'            ; Add 4 to the AD value in W
        andlw   B'11100000'     ; Get the top 3 bits (Result 1)
        movwf   TEMP
        movf    ADC_VALUE, w
        andlw   B'11100000'     ; Get the top 3 bits (Result 2)
        xorwf   TEMP, w         ; Test if the two results are the same
        btfss   ZEROBIT         ; If zero, they're the same
        goto    MainLoop        ; Not the same, no need to change multiplier

        ; We need to change the multiplier
        swapf   ADC_VALUE, f    ; We only need the top 3 bits
        rrf     ADC_VALUE, f
        movlw   B'00000111'     ; Mask unwanted bits
        andwf   ADC_VALUE, w
        movwf   TEMP
        ; Is this the same as the last multiplier index?
        subwf   OLD_MULT_CV, w
        btfsc   ZEROBIT
        goto    MainLoop        ; It's the same, so give up
        ; Use the W value as a lookup to get the multiplier value
        movfw   TEMP
        movwf   OLD_MULT_CV     ; Store this as the Old Mult for next time
        movwf   OLD_MULT_SWITCH ; Switch will increment from here too
        call    GetMultiplier
        movwf   MULTIPLIER      ; Store the value
        ; Multiply the basic Freq Inc by this value
        call    TempoMultiplication
        ; Store the result as the new FREQ_INC
        movfw   MULT_HI
        movwf   FREQ_INC_HI
        movfw   MULT_MID
        movwf   FREQ_INC_MID
        movfw   MULT_LO
        movwf   FREQ_INC_LO
        goto    UpdateFreqIncs

; Update the Output Level CV
OutputLevelCV:
        movlw   b'00010001'     ; AN4, ADC On
        movwf   ADCON0
        call    DoADConversion
        movf    ADC_VALUE, w
        movwf   LEVEL_CV        ; Simply store this one- easy!
        goto    MainLoop

; Update the Phase Distortion CV
PhaseDistortionCV
        movlw   b'00010101'     ; AN5, ADC On
        movwf   ADCON0
        call    DoADConversion
; Limit the range of the DISTORT_CV
PDCVTooLow
        ; Is it less than 4?
        andlw   B'11111100'
        btfss   ZEROBIT ; Is the ADC value 3 or less?
        goto    PDCVTooHigh     ; No, so move on
        movlw   D'4'            ; Yes, so set it to 4 minimum
        goto    StorePDCV

PDCVTooHigh
        ; Is it more than 251?
        xorlw   B'11111111'     ; Invert it
        andlw   B'11111100'
        btfsc   ZEROBIT ; Is the inverted value 3 or less?
        movlw   D'3'            ; Yes, so set it to 252 maximum
        xorlw   B'11111111'     ; Re-invert it to get it back


StorePDCV
        movwf   DISTORT_CV      ; Store the phase distortion value
        ; Update the frequency increments then return to mainloop
        goto    UpdateFreqIncs
```

```
;----------------------------------------
; Update frequency increments (A+B) subroutine
; This is used by FrequencyCV and
; PhaseDistortionCV in the main code loop
;----------------------------------------

UpdateFreqIncs
; Get the FREQ INC value and multiply it by 128. Store in INCTEMP.
        clrf    INCTEMP_LO
        bcf             CARRY
        rrf             FREQ_INC_MID, w
        movwf   INCTEMP_HI
        rrf             FREQ_INC_LO, w
        movwf   INCTEMP_MID
        rrf             INCTEMP_LO, f   ; Shift the extra bit into the low byte

; Divide by the DISTORT_CV value
        movf    DISTORT_CV, w   ; Put the DISTORT_CV value into the divisor
        movwf   DIVISOR
        call    Division        ; Do the division
; Store result as INC A
        movf    NUMBER_HI, w
        movwf   FREQ_INC_A_HI
        movf    NUMBER_MID, w
        movwf   FREQ_INC_A_MID
        movf    NUMBER_LO, w
        movwf   FREQ_INC_A_LO

; Divide by 256-DISTORT_CV value
        comf    DISTORT_CV, w   ; Put 256-DISTORT_CV value into the divisor
        movwf   DIVISOR
        call    Division        ; Do the division
; Store result as INC B
        movf    NUMBER_HI, w
        movwf   FREQ_INC_B_HI
        movf    NUMBER_MID, w
        movwf   FREQ_INC_B_MID
        movf    NUMBER_LO, w
        movwf   FREQ_INC_B_LO
; And that's the lot!
        goto    MainLoop


;***********************************************************
;       Unsigned 24 bit by 8 bit divide routine
;
; Inputs:
;   Dividend  - NUMBER_HI,NUMBER_MID,NUMBER_LO
;   Divisor   - DIVISOR
; Temporary:
;   Counter   - COUNTER
; Output:
;   Quotient  - NUMBER_HI,NUMBER_MID,NUMBER_LO
;   Remainder - REMAIN
;
; Size: 17
; Timing: 342 cycles (including call and return)
;
; This is basically Nikolai Golovchenko's 24 by 16 bit
; divide routine, with some instructions removed to
; optimise it for an 8 bit divide.
; Thanks to Nikolai for the original post.
;
; James Hillman, 2 December 2005
;************************************************************

Division
        clrf    REMAIN          ; Clear remainder
        movlw   D'24'
        movwf   DIVTEMP
GetNumerator
        ; Put INCTEMP (raw freq inc x 128) into the numerator
        movf    INCTEMP_HI, w
        movwf   NUMBER_HI
        movf    INCTEMP_MID, w
        movwf   NUMBER_MID
        movf    INCTEMP_LO, w
        movwf   NUMBER_LO

DivisionLoop
        rlf             NUMBER_LO, w    ; Shift dividend left to move next bit to
remainder
        rlf             NUMBER_MID, f
        rlf             NUMBER_HI, f
        rlf             REMAIN, f       ; Shift carry (next dividend bit) into remainder
        rlf             NUMBER_LO, f    ; Finish shifting the dividend and save carry in
NUMBER_LO.0,
                                ; since REMAIN can be 9 bit long in some cases
                                ; This bit will also serve as the next result bit.

        movf    DIVISOR, w      ; Subtract divisor from 8-bit REMAIN
        subwf   REMAIN, f

;here we also need to take into account the 9th bit of REMAIN, which
;is in NUMBER_LO.0. If we don't have a borrow after subtracting from
;8 bits of REMAIN, then there is no borrow regardless of 9th bit
;value. But, if we have the borrow, then that will depend on 9th bit
;value. If it is 1, then no final borrow will occur. If it is 0, borrow
;will occur. These values match the borrow flag polarity.

        btfsc   BORROW          ;if no borrow after 8 bit subtraction
        bsf     NUMBER_LO, 0    ;then there is no borrow in result. Overwrite
                                ;NUMBER_LO.0 with 1 to indicate no borrow.
                                ;if borrow did occur, NUMBER_LO.0 already
                                ;holds the final borrow value (0-borrow,
                                ;1-no borrow)
        btfss   NUMBER_LO, 0    ;if no borrow after 9-bit subtraction
        addwf   REMAIN, f       ;restore REMAIN. (w contains the value
                                ;subtracted from it previously)
        decfsz  DIVTEMP, f
        goto    DivisionLoop
        return



;----------------------------------------------------------
;       Waveform Lookup Tables
; In actual fact, the straight-line waveforms
; are calculated rather than looked-up.
; It's faster that way.
;----------------------------------------------------------

; Ramp Up
```

```
RampUp:
        movf    PHASE_HI, w     ; Phase accumulator high byte
        goto    ReturnWithValue

; Ramp Down
RampDown:
        comf    PHASE_HI, w     ; Invert Phase accumulator high byte
        goto    ReturnWithValue

; Pulse waveform
Pulse
        btfss   PHASE_HI, 7
        goto    PulseHigh

PulseLow
        movlw   D'0'            ; Low value
        goto    ReturnWithValue

PulseHigh
        movlw   D'255'          ; High value
        goto    ReturnWithValue

; Triangle
Triangle:
; Note that this triangle starts at the highest point
        bcf     CARRY
        rlf     PHASE_MID, w
        rlf     PHASE_HI, w     ; Phase accumulator high byte x 2
        ;Is PHASE_HI higher than 127?
        btfss   PHASE_HI, 7
        xorlw   D'255'          ; Invert it for falling part of wave
        goto    ReturnWithValue

; Sine
Sine:
; Note that this sine starts at the highest point, 90 degrees.
        rlf     PHASE_MID, w
        rlf     PHASE_HI, w     ; Phase accumulator high byte x 2
        btfsc   PHASE_HI, 7     ; If PHASE in second half
        xorlw   D'255'          ; then invert index
        call    GetHalfSine     ; Lookup sine table
        goto    ReturnWithValue


; "Log" Sweep (bottom half of sine wave)
Sweep:
        movf    PHASE_HI, w
        xorlw   D'128'
        call    GetLump         ; Lookup lump table
        xorlw   D'255'          ; Invert it
        movwf   WAVETEMP
        goto    ReturnWithValue

; Lumps (top half of sine wave)
Lumps:
        movf    PHASE_HI, w
        call    GetLump ; Lookup lump table
        movwf   WAVETEMP
        goto    ReturnWithValue


; Return a psuedo-random Sample&Hold waveform
```

```
SampleAndHold:
        movf    RAND_HI, w      ; Use the high byte as our random number
        ; Do we need a new S&H value?
        btfss   SEGMENT_END
        goto    ReturnWithValue         ; No, so return without changing value

NewRandomLevel:
        bcf             SEGMENT_END                     ; This segment end has been dealt
with
        ; Generate a new random point
        clrc
        rrf             RAND_HI, f
        rrf             RAND_LO, f
        movlw   0x00            ; Equivalent to 'No XOR'
        btfss   CARRY
        movlw   0xA1
        xorwf   RAND_HI, f
        xorwf   RAND_LO, f
        movf    RAND_HI, w      ; Use the high byte as our random number
        goto    ReturnWithValue


;----------------------------------------------------------
;       Multiplier Lookup Table
; The table converts from the 0-7 index in W
; to the tap tempo multiplier, which is based on
; half-speed increments.
; This common denominator allows us to calculate all
; the other note ratios as whole numbers.
;----------------------------------------------------------

GetMultiplier:
        movwf   WAVETEMP
        movlw   HIGH MultiplierTable
        movwf   PCLATH
        movfw   WAVETEMP
        addlw   LOW MultiplierTable
        btfsc   CARRY
        incf    PCLATH, f
        movwf   PCL

MultiplierTable:
        retlw   D'1'    ; 1/2 note
        retlw   D'2'    ; 1/4 note
        retlw   D'3'    ; 1/4 note triplet
        retlw   D'4'    ; 1/8th note
        retlw   D'6'    ; 1/8th note triplet
        retlw   D'8'    ; 1/16th note
        retlw   D'2'    ; +2 dummy values for the top of the scale
        retlw   D'2'




;----------------------------------------------------------
;       Waveform Lookup Tables
; The tables contain half a sine wave used for
; the SINE, and a lump waveform used for the SWEEP
; and LUMPS waves.
;----------------------------------------------------------

GetHalfSine:
        movwf   WAVETEMP
        movlw   HIGH SineTable
        movwf   PCLATH
        movfw   WAVETEMP
        addlw   LOW SineTable
        btfsc   CARRY
        incf    PCLATH, f
        movwf   PCL

SineTable:
        dt      D'255', D'255', D'255', D'255', D'255', D'255', D'255', D'254'
        dt      D'254', D'254', D'254', D'254', D'254', D'253', D'253', D'253'
        dt      D'252', D'252', D'252', D'251', D'251', D'251', D'250', D'250'
        dt      D'249', D'249', D'248', D'248', D'247', D'247', D'246', D'246'
        dt      D'245', D'244', D'244', D'243', D'242', D'242', D'241', D'240'
        dt      D'240', D'239', D'238', D'237', D'237', D'236', D'235', D'234'
        dt      D'233', D'232', D'231', D'230', D'230', D'229', D'228', D'227'
        dt      D'226', D'225', D'224', D'223', D'222', D'221', D'219', D'218'

        dt      D'217', D'216', D'215', D'214', D'213', D'212', D'210', D'209'
        dt      D'208', D'207', D'206', D'204', D'203', D'202', D'200', D'199'
        dt      D'198', D'197', D'195', D'194', D'193', D'191', D'190', D'189'
        dt      D'187', D'186', D'184', D'183', D'182', D'180', D'179', D'177'
        dt      D'176', D'174', D'173', D'172', D'170', D'169', D'167', D'166'
        dt      D'164', D'163', D'161', D'160', D'158', D'157', D'155', D'154'
        dt      D'152', D'150', D'149', D'147', D'146', D'144', D'143', D'141'
        dt      D'140', D'138', D'137', D'135', D'133', D'132', D'130', D'129'

        dt      D'127', D'126', D'124', D'123', D'121', D'119', D'118', D'116'
        dt      D'115', D'113', D'112', D'110', D'109', D'107', D'106', D'104'
        dt      D'102', D'101', D'99', D'98', D'96', D'95', D'93', D'92'
        dt      D'90', D'89', D'87', D'86', D'84', D'83', D'82', D'80'
        dt      D'79', D'77', D'76', D'74', D'73', D'72', D'70', D'69'
        dt      D'67', D'66', D'65', D'63', D'62', D'61', D'59', D'58'
        dt      D'57', D'56', D'54', D'53', D'52', D'50', D'49', D'48'
        dt      D'47', D'46', D'44', D'43', D'42', D'41', D'40', D'39'

        dt      D'38', D'37', D'35', D'34', D'33', D'32', D'31', D'30'
        dt      D'29', D'28', D'27', D'26', D'26', D'25', D'24', D'23'
        dt      D'22', D'21', D'20', D'19', D'19', D'18', D'17', D'16'
        dt      D'16', D'15', D'14', D'14', D'13', D'12', D'12', D'11'
        dt      D'10', D'10', D'9', D'9', D'8', D'8', D'7', D'7'
        dt      D'6', D'6', D'5', D'5', D'5', D'4', D'4', D'4'
        dt      D'3', D'3', D'3', D'2', D'2', D'2', D'2', D'2'
        dt      D'2', D'1', D'1', D'1', D'1', D'1', D'1', D'1'



GetLump:
        movwf   WAVETEMP
        movlw   HIGH LumpTable
        movwf   PCLATH
        movfw   WAVETEMP
        addlw   LOW LumpTable
        btfsc   CARRY
        incf    PCLATH, f
        movwf   PCL

LumpTable:
        dt      D'255', D'255', D'255', D'255', D'255', D'255', D'254', D'254'
        dt      D'254', D'253', D'253', D'253', D'252', D'252', D'251', D'251'
        dt      D'250', D'249', D'249', D'248', D'247', D'247', D'246', D'245'
        dt      D'244', D'243', D'242', D'241', D'240', D'239', D'238', D'237'
        dt      D'236', D'234', D'233', D'232', D'230', D'229', D'228', D'226'
        dt      D'225', D'223', D'222', D'220', D'219', D'217', D'215', D'214'
        dt      D'212', D'210', D'208', D'207', D'205', D'203', D'201', D'199'
        dt      D'197', D'195', D'193', D'191', D'189', D'187', D'185', D'182'

        dt      D'180', D'178', D'176', D'173', D'171', D'169', D'166', D'164'
        dt      D'162', D'159', D'157', D'154', D'152', D'149', D'147', D'144'
        dt      D'142', D'139', D'136', D'134', D'131', D'128', D'126', D'123'
        dt      D'120', D'117', D'115', D'112', D'109', D'106', D'103', D'100'
        dt      D'98', D'95', D'92', D'89', D'86', D'83', D'80', D'77'
        dt      D'74', D'71', D'68', D'65', D'62', D'59', D'56', D'53'
        dt      D'50', D'47', D'43', D'40', D'37', D'34', D'31', D'28'
        dt      D'25', D'22', D'19', D'16', D'12', D'9', D'6', D'3'

        dt      D'0', D'3', D'6', D'9', D'13', D'16', D'19', D'22'
        dt      D'25', D'28', D'31', D'34', D'37', D'41', D'44', D'47'
        dt      D'50', D'53', D'56', D'59', D'62', D'65', D'68', D'71'
        dt      D'74', D'77', D'80', D'83', D'86', D'89', D'92', D'95'
        dt      D'98', D'100', D'103', D'106', D'109', D'112', D'115', D'117'
        dt      D'120', D'123', D'126', D'128', D'131', D'134', D'136', D'139'
        dt      D'142', D'144', D'147', D'149', D'152', D'154', D'157', D'159'
        dt      D'162', D'164', D'167', D'169', D'171', D'174', D'176', D'178'

        dt      D'180', D'183', D'185', D'187', D'189', D'191', D'193', D'195'
        dt      D'197', D'199', D'201', D'203', D'205', D'207', D'209', D'210'
        dt      D'212', D'214', D'215', D'217', D'219', D'220', D'222', D'223'
        dt      D'225', D'226', D'228', D'229', D'231', D'232', D'233', D'234'
        dt      D'236', D'237', D'238', D'239', D'240', D'241', D'242', D'243'
        dt      D'244', D'245', D'246', D'247', D'247', D'248', D'249', D'249'
        dt      D'250', D'251', D'251', D'252', D'252', D'253', D'253', D'253'
        dt      D'254', D'254', D'254', D'255', D'255', D'255', D'255', D'255'



;----------------------------------------------------------
;       Frequency Control Lookup Table
; Converts from 0-255 CV input to 16 bit FREQ_INC value
; The main purpose of this is to provide a logaritmic
; (equal octave) response to the LFO FREQ control
; The tables should be called with the index in ADC_VALUE,
; (NOT in the W register), and will return the required value
;----------------------------------------------------------

GetFreqIncHi:
        movlw   HIGH FreqLookupHi
        movwf   PCLATH
        movf    ADC_VALUE, w
        addlw   LOW FreqLookupHi
        btfsc   CARRY
        incf    PCLATH, f
        movwf   PCL

FreqLookupHi:
        dt      D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0',
D'0', D'0', D'0', D'0'
        dt      D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0',
D'0', D'0', D'0', D'0'
        dt      D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0',
D'0', D'0', D'0', D'0'
        dt      D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0',
D'0', D'0', D'0', D'0'
```

```
        dt      D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0',
D'0', D'0', D'0', D'0'
        dt      D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0',
D'0', D'0', D'0', D'0'
        dt      D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0', D'0',
D'0', D'0', D'0', D'0'
        dt      D'0', D'0', D'0', D'1', D'1', D'1', D'1', D'1', D'1', D'1', D'1', D'1',
D'1', D'1', D'1', D'1'
        dt      D'1', D'1', D'1', D'1', D'1', D'1', D'1', D'1', D'1', D'1', D'1', D'1',
D'1', D'1', D'1', D'1'
        dt      D'1', D'1', D'1', D'2', D'2', D'2', D'2', D'2', D'2', D'2', D'2', D'2',
D'2', D'2', D'2', D'2'
        dt      D'2', D'2', D'2', D'2', D'2', D'2', D'3', D'3', D'3', D'3', D'3', D'3',
D'3', D'3', D'3', D'3'
        dt      D'3', D'3', D'3', D'4', D'4', D'4', D'4', D'4', D'4', D'4', D'4', D'4',
D'4', D'5', D'5', D'5'
        dt      D'5', D'5', D'5', D'5', D'5', D'5', D'6', D'6', D'6', D'6', D'6', D'6',
D'6', D'7', D'7', D'7'
        dt      D'7', D'7', D'7', D'8', D'8', D'8', D'8', D'8', D'9', D'9', D'9', D'9',
D'9', D'10', D'10', D'10'
        dt      D'10', D'10', D'11', D'11', D'11', D'11', D'12', D'12', D'12', D'13',
D'13', D'13', D'13', D'14', D'14', D'14'
        dt      D'15', D'15', D'15', D'16', D'16', D'16', D'17', D'17', D'18', D'18',
D'18', D'19', D'19', D'20', D'20', D'21'

GetFreqIncLo:
        movlw   HIGH FreqLookupLo
        movwf   PCLATH
        movf    ADC_VALUE, w    ; Get index directly from ADC_VALUE
        addlw   LOW FreqLookupLo
        btfsc   CARRY
        incf    PCLATH, f
        movwf   PCL

FreqLookupLo:
        dt      D'22', D'22', D'22', D'23', D'23', D'24', D'24', D'25', D'26', D'26',
D'27', D'27', D'28', D'29', D'29', D'30'
        dt      D'30', D'31', D'32', D'32', D'33', D'34', D'35', D'35', D'36', D'37',
D'38', D'39', D'39', D'40', D'41', D'42'
        dt      D'43', D'44', D'45', D'46', D'47', D'48', D'49', D'50', D'51', D'52',
D'53', D'55', D'56', D'57', D'58', D'60'
        dt      D'61', D'62', D'64', D'65', D'66', D'68', D'69', D'71', D'72', D'74',
D'76', D'77', D'79', D'81', D'82', D'84'
        dt      D'86', D'88', D'90', D'92', D'94', D'96', D'98', D'100', D'102', D'105',
D'107', D'109', D'112', D'114', D'117', D'119'
        dt      D'122', D'124', D'127', D'130', D'133', D'136', D'139', D'142', D'145',
D'148', D'151', D'154', D'158', D'161', D'165', D'168'
        dt      D'172', D'176', D'180', D'184', D'188', D'192', D'196', D'200', D'205',
D'209', D'214', D'218', D'223', D'228', D'233', D'238'
        dt      D'243', D'249', D'254', D'4', D'9', D'15', D'21', D'27', D'33', D'40',
D'46', D'53', D'60', D'66', D'74', D'81'
        dt      D'88', D'96', D'103', D'111', D'119', D'128', D'136', D'144', D'153',
D'162', D'171', D'181', D'190', D'200', D'210', D'220'
        dt      D'231', D'241', D'252', D'7', D'19', D'30', D'42', D'54', D'67', D'79',
D'92', D'106', D'119', D'133', D'147', D'162'
        dt      D'176', D'191', D'207', D'223', D'239', D'255', D'16', D'33', D'51',
D'68', D'87', D'105', D'125', D'144', D'164', D'185'
        dt      D'205', D'227', D'248', D'15', D'37', D'61', D'84', D'109', D'134',
D'159', D'185', D'211', D'238', D'10', D'38', D'67'
        dt      D'97', D'127', D'158', D'189', D'221', D'254', D'32', D'66', D'101',
D'137', D'174', D'211', D'249', D'32', D'72', D'113'
        dt      D'155', D'197', D'241', D'30', D'75', D'121', D'169', D'218', D'11',
D'62', D'114', D'167', D'221', D'20', D'76', D'134'
        dt      D'193', D'253', D'59', D'122', D'186', D'252', D'63', D'132', D'202',
D'18', D'91', D'166', D'242', D'65', D'145', D'226'
        dt      D'54', D'139', D'226', D'59', D'150', D'243', D'82', D'179', D'22',
D'124', D'227', D'77', D'185', D'40', D'153', D'12'
; We never reach here
        end
```