Replacement M32 Module for Wavetek 3000 Signal Generator

Recently I picked up a fixer-upper Wavetek 3000 signal generator at a flea market at a good price because it had a known lock problem. My HP generator only covers up to 13MHz, and the Wavetek would come in handy for those higher HF and VHF projects.

After fixing the lock issue (bad voltage regulator in M31), I could read the correct frequency on the counter and the output levels were pretty much where they should be. When listening to the signal output on the HF receiver, I couldn't find a tone to zero-beat, just an increase in noise floor. On AM It was the same, just a tunable noise hump. VHF FM, the same.

There have been many reports of op-amp, voltage regulator, and electrolytic capacitor failure in these old equipment. The op-amps were easy to replace, and suspicious capacitors were changed out. The noise still persisted without change.

One by one the VCO's were checked on the spectrum analyzer and the M32 module was REALLY noisy. I wasn't expecting HP like signal purity, but this was certainly outside of the specs. By applying a clean DC voltage to the varactor tune line, I did notice that the noise was less (still awful) at low tune voltages and more at higher tune voltages. The varactor was replaced with a fixed cap and the output signal was clean. I scrounged around and found a varactor that covered the same frequency range with a fairly close voltage range and boxed everything back up.

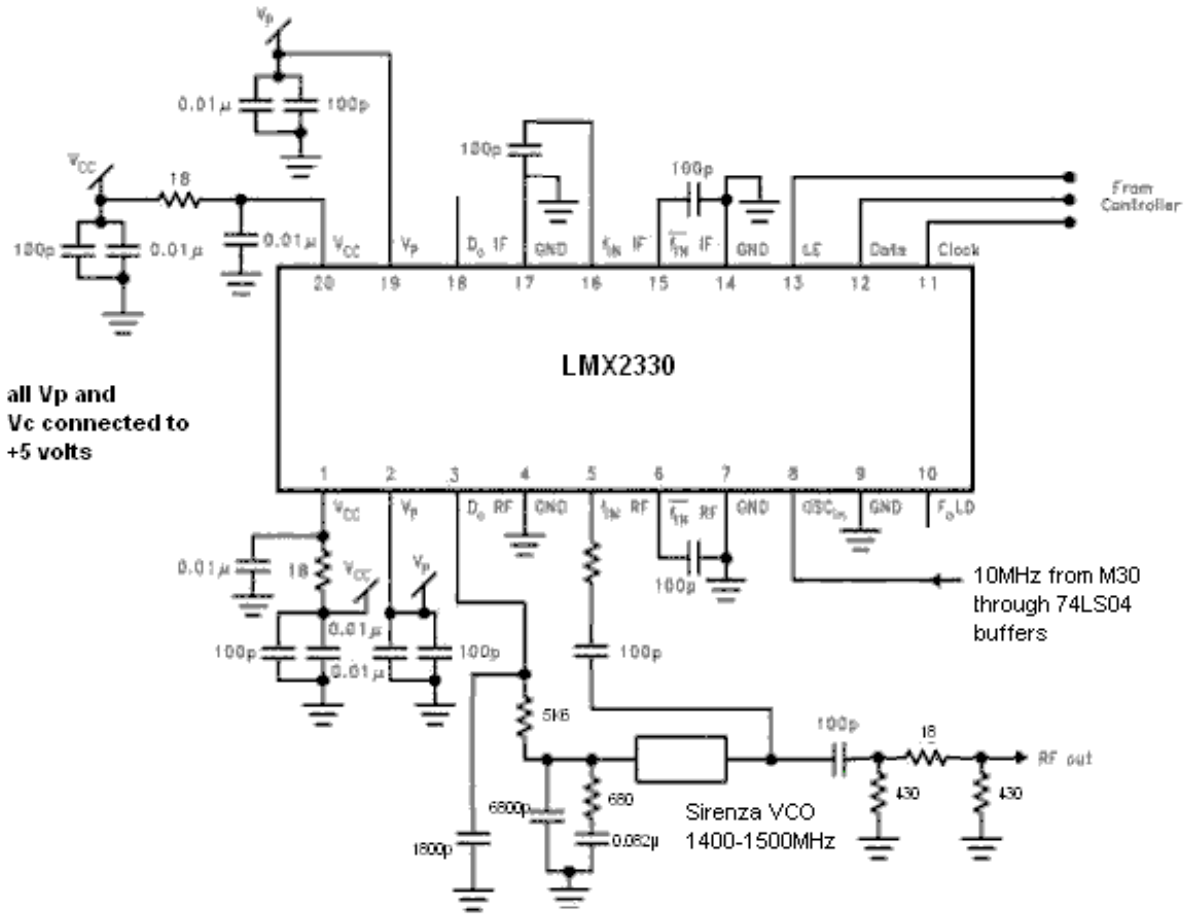
The result was much better, but still sounded raspy and would break lock on fast or large frequency changes. Adjusting the inductor (a bent wire) tap inside that rat's nest of point-to-point wiring is not the easiest task. The whole thing went on the shelf for some "rest".

Some idle time thinking on what is needed lead to a decision to replace the functionality with something simpler, more robust and cleaner. There are too many years on this, too many mixers and level shifters, and not enough filtering, to give nice clean results. All that is needed are simple 1 MHz steps, from 1448 to 1487MHz, controlled by the front panel BCD switches. Should be a simple job for a simple PLL circuit.

I scrounged around for parts and found a VCO with the right range from an old abandoned project, an LMX2330 synthesizer chip from an old cell phone, and a 16F84 PIC. The LMX goes up to 3GHz and is a dual PLL chip; any PLL (like the LMX for ADF series) that covers at least 1.5GHz would do fine, and a single would be fine as this project doesn't use the dual feature. The PIC code is included here and should be easily translatable to almost any PIC with at least 10 I/O lines available (7 for the BCD switches and 3 for the PLL control lines).

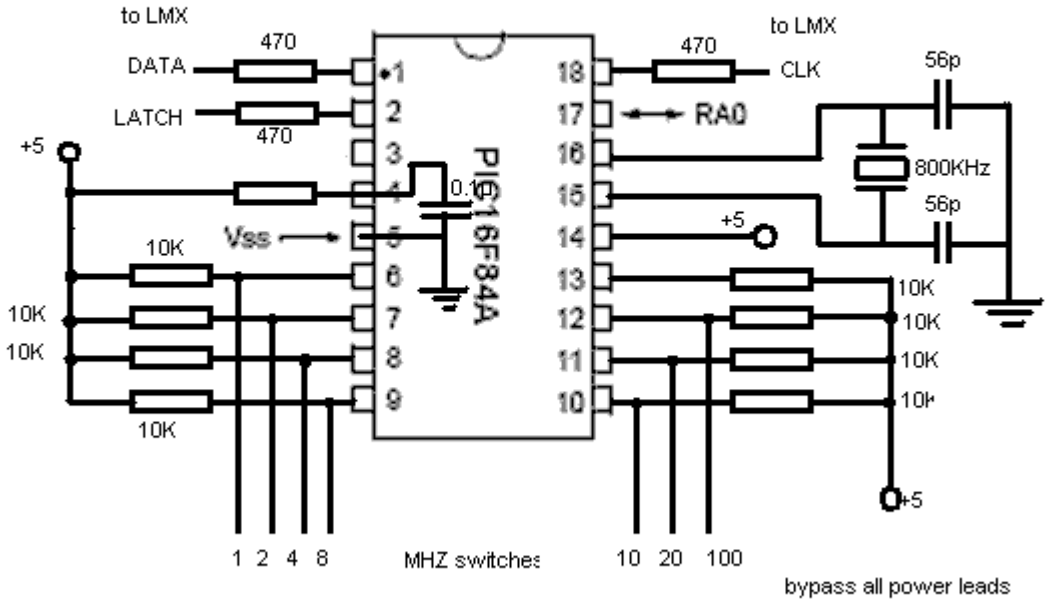
Theory:
Standard PLL circuit from the VCO through the synthesizer chip. I used NSC's EasyPLL web software to calculate the loop values. The synthesizer section was tested with NSC's CodeLoader software which allows loading the PLL via a PC parallel port. The output was clean, stayed in lock throughout the entire range of the VCO, and really didn't start to get cranky until way outside the loop design parameters. The loop is designed for a 1MHz reference and 10KHz

bandwidth. One can see the lock stability and reference sideband suppression change as the reference is varied from 50KHz to 5MHz with the same output frequency on the same loop values. It's pretty neat to be able to change all these parameters from the computer screen with the NSC demo software. This was my first time using it. The unused IF section of the dual PLL is programmed to power-down. The 10MHz reference input is buffered and is pulled from the M30 reference module unused test jack. The buffer circuit is not shown; it is just using some inverter gates from a 74LS04. The 1MHz output can be used but with a lower reference frequency step as the minimum R value is 3 for the LMX. I wanted to keep step size equal to the channel spacing to ease the programming requirements (though a 250KHz or 125KHz reference would just require some addition bit shifting in the program). The lock indicator functionality was not used, as if you give this power and a 10MHz reference, it locks.



The PIC code is fairly simple (short), but briefly it does this: initializes the registers, checks to see if the module is plugged in, and if not, sets a default for testing so that the PLL doesn't try to lock on an invalid BDC code. Then it checks the input ports from the BCD switches, converts the BCD code to the binary needed to program the PLL, loads this to the PLL, and then goes back to read the switches again looking for a change. If it changes, the load process repeats; if not it just loops. I use the watch dog timer (never have before) just in case the program gets scrambled from transmitter testing on the bench or such. I use a low frequency ceramic resonator because this tends to be less noisy than the external RC clock source and a higher

frequency crystal is not needed for speed or stability. Anything from a 32KHz clock to a 4MHz crystal would work as well, as the LMX can handle a fairly fast serial input.



It works, does what it's intended to do, is much simpler (and cleaner) than what it replaced. It's fast enough that you can see the signal lock on a new frequency due to the switch bounce as the BCD switches are changed (maybe this is too fast, but I wasn't going to spend time putting delays into the code because it's a feature that really isn't operationally needed).

This is not intended to be an exact construction project, but is offered more as a reference to stir some ideas for needs you may have that are similar. None of the parts are value critical, though with PLL's you don't want to stray too far with the loop parameters. If changing the VCO tuning sensitivity, PLL chip, reference frequency, or N values; I'd strongly recommend using one of the loop filter tools to select the component values.

Enjoy

73's de WA2SCL

```
; Replacement M32 module for Wavetek 3000 Signal Generator
; by WA2SCL  28Feb2012

; input selector binary shift
; 100   20    10   input  output
;  0     0     0    00X    00
;  0     0     1    01X    01
;  0     1     0    02X    10
;  0     1     1    03X    11
;  1     0     0    10X    10
;  1     0     1    11X    11
;  1     1     0    12X    00
;  1     1     1    13X    10

; inputs - pin functions
; RB0 = 1          (6)
; RB1 = 2          (7)
; RB2 = 4          (8)
; RB3 = 8          (9)
; RB4 = 10         (10)
; RB5 = 20         (11)
; RB6 = 100        (12)
; outputs
; RA1 = CLK      (18)
; RA2 = DATA   (1)
; RA3 = LE        (2)

    LIST p=16F84A                   ; tell assembler what chip we are using
    #include "p16F84a.inc"
```

```
        __config _CP_OFF & _WDT_ON & _PWRTE_ON & _XT_OSC  ;set WDT_ON for production

        errorlevel          -302

W       equ    0
F       equ    1
CLK1    equ    1                ; set to PORTA:1
DAT1    equ    2                ; set to A:2, etc
LE1     equ    3

OldBCD  equ    020h
NewBCD  equ    021h
Tens    equ    022h
Units   equ    023h
Bcounter equ   024h
Acounter equ   025h
SwBin   equ    026h
temp    equ    027h
HighB   equ    030h
MidB    equ    031h
LowB    equ    032h

   org 0
   nop
   goto    Init

   org 5
Offset_table
   addlw    1
```

```
    addwf     PCL, W
    movwf     PCL
    retlw     B'00000000'      ; 00X
    retlw     B'00000001'      ; 01X
    retlw     B'00000010'      ; 02X
    retlw     B'00000011'      ; 03X
    retlw     B'00000010'      ; 10X
    retlw     B'00000011'      ; 11X
    retlw     B'00000000'      ; 12X
    retlw     B'00000001'      ; 13X

BCD_Table
    addlw     1
    addwf     PCL, W
    movwf     PCL
    retlw     B'00000000'      ; 00X
    retlw     B'00001010'      ; 01X
    retlw     B'00010100'      ; 02X
    retlw     B'00011110'      ; 03X

BCD2Bin
     ; converts the BCD switch setting to binary
    bcf       STATUS, C
    movf      Tens, W
    call      BCD_Table
    addwf     Units, W     ; now W has the binary switch setting range (00 to 39)
    movwf     SwBin        ; save switch binary
    addlw     0x08
    andlw     B'00011111'  ; mask off A counter value
```

```
    movwf     Acounter     ; this is the swallow counter value
    movf      SwBin, W     ; get it back
    addlw     .232
    btfsc     STATUS, C    ; if clear, then A = 45
    goto      Set46
    movlw     .45
    movwf     Bcounter     ; load B counter with correct value
    goto      ExitBCD
Set46
    movlw     .46
    movwf     Bcounter
ExitBCD
    return

CounterN
; get divisor ready for PLL format
    clrf      HighB
    movfw     Acounter
    movwf     LowB
    movfw     Bcounter
    movwf     MidB
    rlf       LowB, 1
    rlf       LowB, 1   ; move over to make room for control bits
    bsf       LowB, 0
    bsf       LowB, 1   ; control bits  1,1 for RF N
    bcf       STATUS, C
    rlf       MidB, 1
    return              ; ready to load - right justitifed data
```

```
LoadPLL
; sends data to PLL
        btfss   HighB,5             ; Bit 22
        call        zero1
        btfsc   HighB,5
        call        one1
        btfss   HighB,4             ; Bit 21
        call        zero1
        btfsc   HighB,4
        call        one1
        btfss   HighB,3                 ; Bit 20
        call        zero1
        btfsc   HighB,3
        call        one1
        btfss   HighB,2                 ; Bit 19
        call        zero1
        btfsc   HighB,2
        call        one1
        btfss   HighB,1                 ; Bit 18
        call        zero1
        btfsc   HighB,1
        call        one1
        btfss   HighB,0                 ; Bit 17
        call        zero1
        btfsc   HighB,0
        call        one1
        btfss   MidB,7                  ; Bit 16
        call        zero1
        btfsc   MidB,7
```

```
call    one1
btfss   MidB,6          ; Bit 15
call    zero1
btfsc   MidB,6
call    one1
btfss   MidB,5          ; Bit 14
call    zero1
btfsc   MidB,5
call    one1
btfss   MidB,4          ; Bit 13
call    zero1
btfsc   MidB,4
call    one1
btfss   MidB,3          ; Bit 12
call    zero1
btfsc   MidB,3
call    one1
btfss   MidB,2          ; Bit 11
call    zero1
btfsc   MidB,2
call    one1
btfss   MidB,1          ; Bit 10
call    zero1
btfsc   MidB,1
call    one1
btfss   MidB,0          ; Bit 9
call    zero1
btfsc   MidB,0
call    one1
```

```
btfss   LowB,7          ; Bit 8
call    zero1
btfsc   LowB,7
call    one1
btfss   LowB,6          ; Bit 7
call    zero1
btfsc   LowB,6
call    one1
btfss   LowB,5          ; Bit 6
call    zero1
btfsc   LowB,5
call    one1
btfss   LowB,4          ; Bit 5
call    zero1
btfsc   LowB,4
call    one1
btfss   LowB,3          ; Bit 4
call    zero1
btfsc   LowB,3
call    one1
btfss   LowB,2          ; Bit 3
call    zero1
btfsc   LowB,2
call    one1
btfss   LowB,1          ; Bit 2
call    zero1
btfsc   LowB,1
call    one1
btfss   LowB,0          ; Bit 1
```

```
        call            zero1
        btfsc   LowB,0
        call            one1
;
load1
        call    latch_en1
        return

; pings the LE line
latch_en1
            bsf     PORTA, LE1
            bcf     PORTA, LE1
            return

; Subroutines to send 0 and 1
zero1
            bcf     PORTA, DAT1
            bsf     PORTA, CLK1
            bcf     PORTA, CLK1
    return
;
one1
            bsf     PORTA, DAT1
            bsf     PORTA, CLK1
            bcf     PORTA, CLK1
            bcf     PORTA, DAT1
    return

PlugCheck   ; deafults to 01 if not plugged
```

```
    movwf     temp          ; w has and will have the switch
    incf      temp, F
    btfsc     STATUS, Z     ; if didn't overflow, is valid
    movlw     01h           ; overflowed, so default
    bcf       STATUS, Z     ; not needed but avoids errors in case
    return                  ; w is desired switch state

; start code for program initialization
Init
    clrf      STATUS        ; Do initialization, Select bank 0
    clrf      INTCON        ; Clear int-flags, Disable interrupts
    clrf      PCLATH        ; Keep in lower 2KByte
    bsf       STATUS,RP0            ; bank1
    movlw     0xFF
    movwf     TRISB         ; set portb as input 0-7
    clrf      TRISA         ; set porta as output 0-5
    movlw     B'11111110'
    movwf     OPTION_REG    ; Disable PORT_B pull-ups, TMR0 to WDT
    bcf       STATUS,RP0
    clrf      OldBCD
    clrf      NewBCD
    clrf      Units
    clrf      Tens
    clrf      Acounter
    clrf      Bcounter
    clrf      SwBin

    clrf      PORTA       ; have RA1-3 (3 bits)
    clrf      PORTB       ; have RB0-6 (7 bits)
```

```
    clrwdt



    comf      PORTB, W      ; save the BCD values on startup
; flip the bits as the input is ground active
    call      PlugCheck     ; check if plugged in!
    movwf     OldBCD

; initialize the IF R and N regs as well as RF R reg
    movlw     28h
    movwf     LowB
    movlw     00h
    movwf     MidB
    movlw     0Ah
    movwf     HighB
    call      LoadPLL          ; load the IF R register
    movlw     01h
    movwf     LowB
    movlw     0FEh
    movwf     MidB
    movlw     2Fh
    movwf     HighB
    call      LoadPLL          ; load the IF N register
    movlw     2Ah
    movwf     LowB
    movlw     00h
    movwf     MidB
    movlw     06h
```

```
    movwf     HighB
    call      LoadPLL          ; load the RF R register
    call      Begin            ; now get the RF N register
    goto      Loop1

Begin
     ; prepares the switch value and send to PLL
    swapf     OldBCD, W
    andlw     B'00000111'  ; mask off units BCD, put it W
    call      Offset_table ; now W has the tens BCD ready for PLL
    movwf     Tens
    movf      OldBCD, W
    andlw     B'00001111'  ; now units is OK in W
    movwf     Units
    call      BCD2Bin      ; puts the binary value in W
    call      CounterN     ; stores the counter A & B (swallow & N) values in W
    call      LoadPLL
    return

Loop1
    clrwdt                 ; reset the WDT
    comf      PORTB, W       ; get the switch setting
    call      PlugCheck
    movwf     NewBCD         ; save newBCD, stays in W
    subwf     OldBCD, W      ; if the difference is 0, then no change
    btfsc     STATUS, Z
    goto      Loop1          ; keep checking
    movf      NewBCD, W
    movwf     OldBCD         ; save the new value as the old (current)
```

```
call      Begin
goto      Loop1


end
```