

By Jan Skirrow

Building a Direct Digital Synthesis Oscillator

My interest in microwave frequencies has resulted in several projects to convert surplus Qualcomm phase-locked oscillators for use as local oscillators in various transverters.

The boards have proven very useful to those who manage to acquire them. Each of the Qualcomm boards uses a TCXO (sometimes mounted on the board, and sometimes not) as a reference for a phase locked oscillator based on a proprietary Qualcomm PLL chip.

This chip can be controlled with a parallel instruction word - which makes conversion of the board to other frequencies fairly easy. Some boards are hardwired for a single frequency, while other versions bring the control lines out to a connector.

With the hardwired boards it can be very difficult to change the frequency again, once the necessary surgery on the PLL chip has been done once.

More detail on these boards and conversions can be found on the San Bernardino Microwave Group's website under "*Technical Papers & Software from the San Diego Microwave Group*" - <http://www.ham-radio.com/sbms/> or at <http://www.ham-radio.com/sbms/sd/>

I've looked at ways of making the Qualcomm boards somewhat tuneable by means of a tuneable replacement for the usual TCXO, or to replace them completely using the newer direct digital synthesis chips that are becoming more available at a reasonable price.

Curtis Preuss, WB2V's DDS VFO

The July 1997 issue of QEX, published by the ARRL, contains a construction article for a DDS oscillator based on the Analog Devices

AD9850 chip. The project was intended mostly to demonstrate the feasibility of DDS for homebrewers, and to provide a small, stable VFO for various QRP and similar projects.

The AD9850 can be clocked to a maximum frequency of 125MHz, which means a maximum useable output frequency of about half that. This requires a stable 125MHz reference oscillator, which can be difficult to find. Preuss's design used a 66.6667MHz crystal oscillator as the reference drive for the DDS chip, which limited the maximum output frequency to about 30MHz. He further reduced this through the control software to suit his needs.

Analog Devices provides a circuit for a stable crystal oscillator that allows the AD9850 to be clocked near its maximum frequency.

The raw output of the AD9850 contains not only the desired fundamental frequency, but also the sum and difference of it with the clocking frequency. So with appropriate output filters, this chip can perform well over quite a wide range of frequencies. While this can be an advantage in some situations, care is needed in selecting the reference oscillator so that sum and difference frequencies don't appear within the bandpass of the output filter.

Duplicating the DDS VFO

I'm not going to repeat the excellent discussion of the circuit contained in Preuss's article, so I will assume you have read it. But I thought it might be useful to include some information on the parts of the construction that I did somewhat differently, or had problems with, and to suggest some follow-up projects. Others have also extended this design - a Google on AD9850 will find these for you.

I decided to produce a professional quality printed circuit board that could serve as a test bed for further projects. Analog Devices sug-



gests using a multi-layer board with separate ground and power planes, in addition to the signal layers. Preuss used a board having all the traces on one side, with the backside used as a ground plane. I used double-sided (because it is easy to do with the software I have) but maintained a ground plane under the RF section of the board.

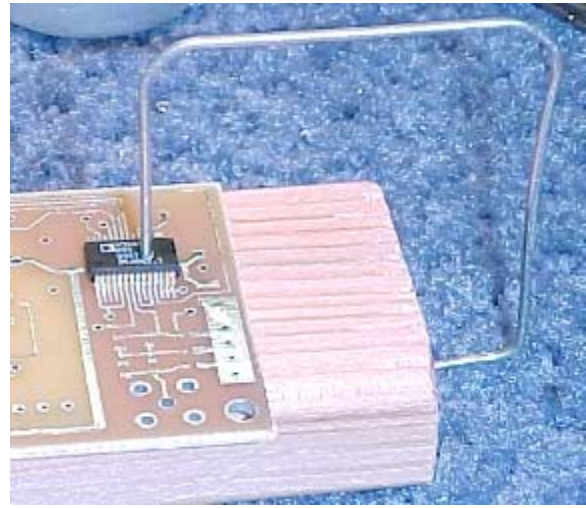
The picture above shows the board and display laid out to facilitate access to the PIC. There are more pictures at the end of this article.

I used a 14 pin SIP pattern for the connection to the LCD display so that I could hang the DDS board directly from the display using stiff bare wire. The SIP pattern matches exactly the connection pins on the LCD display.

I had a different optical encoder in my junk box than the one Preuss called for. Thus my board has a 4 pin connector that will connect to any four wire encoder, and a separate 2 pin SIP for connection to a push button that enables a higher speed tuning rate.

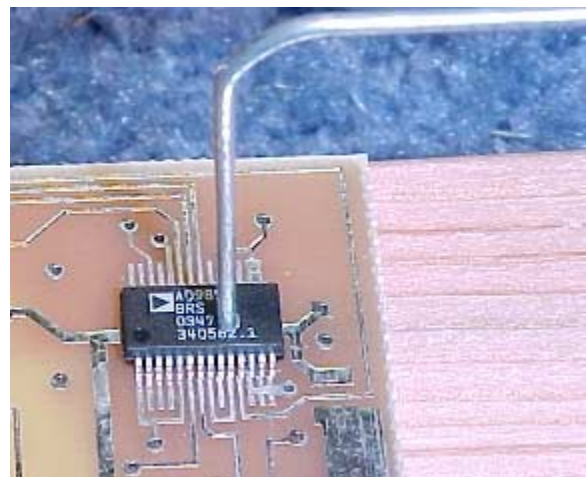
Building the DDS Board

The only real challenge in construction is attaching the AD9850. It has a 28 pin SSOP



package - which means that it can be very difficult to solder by hand.

I use two methods to solder SM components. For things like the AD9850, I have a very simple holding jig made from a large paper clip. This is shown in the pictures above and below. For best results, and for smaller parts, it is best to epoxy the clip into the wood base, and put something on the other end to keep the part from easily slipping out from under. A tiny bit of rubber works well, but



usually isn't necessary for larger parts.

I use a very small 1/16" tip on my soldering iron, and 15mil wire solder. I also use a rosin flux pen to lightly coat the traces and the SSOP pins so that nothing will prolong the contact

time between soldering iron and part. There is no question that a very steady hand and strong working lights are necessary as well! Some people use a magnifying visor, but I find this makes eye-hand coordination more difficult.

With the chip very carefully aligned to the pcb pattern (making sure you have it right way round!), tack one corner. If the chip stays put, then move to the other corners. If not carefully reheat and quickly realign the chip.

I find it helpful to solder by placing the iron on the trace, with the body of the iron aligned along the trace so that there is less chance of accidentally soldering adjacent pins together - I've never been much good with wicking up excess solder! Then the soldering iron is slid carefully towards the SSOP pin and a very small amount of solder applied when they are close together. Repeat this for all pins.

It can be a real challenge to determine if the solder joints are good. The solder is very shiny, and the parts very small. I use a 10x microscope to inspect the pins. By positioning the board so that you are looking at the line between pin and trace, with your work light set to illuminate this fairly strongly, you should see the characteristic solder "dimple" clearly showing flow on both the trace and pin. If it clearly is not soldered (lumpy or an obvious break) carefully reheat, but add little or no additional solder. If solder builds up and you short two adjacent pins, it will be difficult to clean up without damaging the chip.

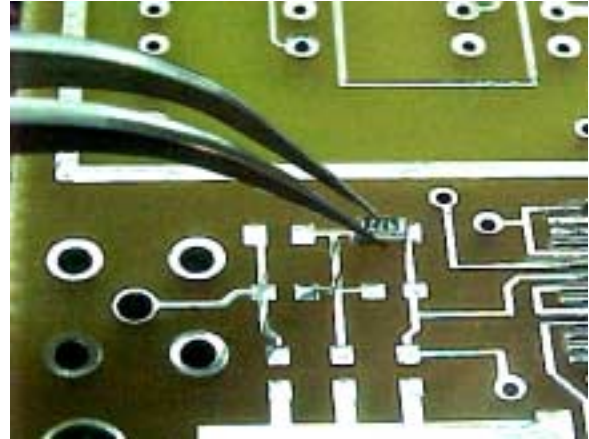
You can use a similar approach for the other small SM parts. Because I already had them, I used some caps in the 0603 package - which is very small. But these are actually easier to solder than the SSOP providing you've left enough room on the pcb to get the soldering iron into the trace.

For these, I place a small amount of solder on the pads the part is to be attached to - just enough to create a small rise in the pad. Then, holding the part with very fine tweezers (I really like the angled version shown in the picture below) align the chip over the pads, place one end firmly on its pad and apply the iron briefly. The solder should flow almost instantly and the joint is done. If the part has

slipped a bit, reheat and realign with the tweezers. Once one end is done, the other end is easy.

On the board shown in the pictures, I didn't have inductors that fit the pad layout I used, so I had to squeeze these more than is desirable. The layout files with this TechTalk have been re-routed to give more working space on the board.

The only other difference between the board



shown in the pictures, and that in the files is that I original placed a SMA connector pattern on the board. But the board will always be mounted in or on something else, so I have deleted the connector and replaced it with two pads that a piece of small coax can be soldered to.

The Software

The original Preuss board used a Microchip PIC 1654 controller to manage the LCD display and DDS chip. His software was later ported to a PIC 16C84 by G. Heron, N2APB. I used a PIC16F84A, which worked fine with the later version of the software without change.

However, I wanted my oscillator to be as accurate as possible. Any commercial crystal oscillator used for the reference frequency will be off the nominal value by some unknown amount. Thus, the LCD display will not indicate the correct output frequency because it reflects the control word to the DDS rather than a measurement of the actual frequency. In other words, it shows the frequency you want, but not necessarily what you get.

I wanted to correct for this, and also to use a 66.0MHz oscillator that I already had. This meant changing the software. And in case you decide to do something similar, I will describe what's needed.

Calculating the Digit Weights

The DDS chip uses an algorithm to form the 32bit frequency control word needed to set the output frequency. This control word is created by the software by using "weighting factors" for each LCD digit. The why isn't important, but the how is, and none of the documentation I could find really described this adequately. Maybe I'm the only one that doesn't find it obvious!

The DDS output is calculated from the expression supplied by Analog Devices:

$$F_{out} = (\Delta\text{Phase} \times \text{CLKIN}) / 2^{32}$$

Where F_{out} is the output frequency in MHz, CLKIN is the reference oscillator frequency in MHz. And ΔPhase is the value of the 32 bit control word the DDS requires.

Preuss then relates this to the digit weights by:

$$\Delta\text{Phase} = \sum_n \text{LCD_Digit} \times \text{Digit_Weight}$$

Where n is the number of digits - seven in this case.

Preuss provides the values in the QEX article. These are repeated below in Table 1. His values were calculated for a 66.6666MHz reference oscillator.

Table 1 - Digit Weights

| <i>LCD_Digit</i> | <i>Digit Weight (hex) 66.6666MHz Ref</i> | <i>Digit Weight hex 66.00051MHz Ref</i> |
|------------------|--|---|
| 10's | 284 | 28B |
| 100's | 192A | 196B |
| 1k | FBA9 | FE33 |
| 10k | 9D49B | 9EDFC |
| 100k | 624E13 | 634BD4 |
| 1M | 3D70CC1 | 3E0F647 |
| 10M | 26667F90 | 26C99EC8 |

What I did was to first calculate ΔPhase for the reference crystal frequency I used and an output frequency of 10MHz. This number is the decimal version digit weight for the 10M digit. By dividing this number by 10, and by 10 again, and so on, the digit weight for each digit is calculated. If you measure the actual frequency of the crystal oscillator you use, once it is 'warmed up', the final unit will be very accurate.

To work in the software, these numbers have to be converted to hexadecimal. If you have a calculator that does this, it is very simple. If you do not (I don't) then you must convert manually. For those of you who have forgotten how to convert between numbers of different base (I sure had!) then here is how you do it.

My reference crystal oscillator measured 66.00051MHz after an hour running. So, I first calculated ΔPhase :

$$\Delta\text{Phase} = (F_{out} \times 2^{32}) / \text{CLKIN} = 650747592.102$$

Then, to convert this digit weight for the 10M digit to hex, divide the number by 16. Put the part before the decimal in the first column in the table on the last page, and the part after the decimal in the second column. Multiply this last number by 16 to get the remainder. Round it up or down as appropriate. This is a hex digit, but remember that hexadecimal is 0 1 2 3 4 5 6 7 8 9 A B C D E F. So if the remainder is 13, then the digit is actually D. If the remainder is 8, the hex digit is 8. Repeat this process until the number before the decimal is less than 16 - this then is the remainder for the last hex digit.

Table 2 shows this calculation for the 10M weight only. The remaining digits are calculated the same way, always starting with a number that is ΔPhase divided by ten successively. So for example, the starting number for the 1M digit weight calculation will be 65074759.210 and for the last digit it will be 650.748.

In each case, the first hex digit calculated is the least significant digit. So the final digit weight for each is as shown in Table 2.

Inserting Digit Weights in the Software

See the link to the software listing on the last page. There is a subroutine called MULT on page 8 of the listing. It contains a series of instructions to load the digit weights for each digit into registers, and then to send them to the DDS. The hex digits calculated and shown in the table at the end of this article are entered as 2 character bytes. Comparing the table with the software will make this fairly obvious.

It is possible that when you calculate the digit weights for your particular reference frequency, you may end up with a different number of digits than shown here. You will need to modify the program.

Note that each digit has four memory locations for storing digits. These are OP1+0, OP1+1, OP1+2 and OP1+3. If you have 8 hex digits (four pairs) they are loaded into these memory locations in pairs, starting with the LSB, using instructions like:

```
movlw 0xYY
movwf OP1+0
movlw 0xZZ
movwf OP1+1 etc.
```

where YY and ZZ are two hex digits. If you only have 5 or 6 hex digits, you load what you have (in pairs - including a leading zero if you have a single digit in the last spot). Then you must clear the unused memory location (who knows what's been left in there by another part of the program!) This is an instruction like:

```
clrf OP1+2
clrf OP1+3
```

In other words, each digit is loaded by placing hex bytes (in pairs) into the memory locations, and clearing unused locations.

If you aren't a programmer, this may be hard to follow, but compare the listings with the above and it should come clear.

Programming the PIC

You do have to download your software into the PIC. That requires a programming module of some kind. I won't go into that here, but lots of info is available on the Microchip website <http://www.microchip.com/>. Note that the PIC used here was chosen because it is a flash memory unit, that can be repro-

grammed indefinitely, and does not require a UV EPROM eraser.

Other Software Issues

The listing is for MPASM assembler, as used by the PIC maker Microchip. If you use a different assembler language the listing will have to be changed.

The Preuss program limits the DDS VFO to a range of 100kHz to 20MHz. However, the DDS chip with a 60MHz reference oscillator will provide usable output to 30MHz.

On page 3 of the software listing, there is a routine named BCDinc This routine increments the LCD as called for by the optical encoder, and checks that the result is below the maximum frequency - 20MHz in this case. If it is attempting to go over this limit, the increment request is ignored, and the display (and the DDS) kept at 20MHz.

This routine can be changed easily to set 30MHz as the maximum frequency. To pick a limit other than 10, 20 or 30 requires more extensive changes (and there isn't a lot of unused space in the PIC used here).

Similarly, the routine called BCDdec on page 4 decrements the LCD display and tests if the lower limit is exceeded. If such an attempt is being made, the display will hold at 100kHz.

When power is applied to the DDS VFO, the LCD display and DDS are turned on and initialized. The PIC is initialized, and a default turn-on frequency of 10MHz is set. In the listing, the routine called Start on page 10 does this. The section initializes variables and loads the start-up frequency in LCD_Dat+0, LCD_Dat+1 and so on. The first location is the 10M digit. You can set these to any start-up value you want within the limits of the min/max set above.

Results

The unit works very well, with an output frequency error that is smaller than the display resolution. The output sine wave is very good, especially below 20MHz, as the filter in the Preuss design eliminates the glitches resulting from the DDS core, and the sum and difference frequencies mentioned previously.

Future Projects

The basic Preuss design has been extended by others to include temperature stabilization. Other things could be done, although the PIC used here is probably too small to handle much more. Other PICs are available that would allow considerably more to be done.

However, this design works well for my basic need to have a tuneable replacement for the 10MHz TCXOs used in my local oscillator projects.

The next step is to eliminate the Qualcomm boards completely. While these are marvellous to have, they aren't easy to find, and will get nothing but rarer!

The Analog Devices AD9851 is pin for pin compatible with the AD9850. But it clocks to 180MHz, and has a built-in reference frequency multiplier that gives full clocking speed with a 30MHz reference oscillator.

This allows output frequencies up to 90MHz. By taking advantage of the fact that sum and difference frequencies are also present in the raw DDS output (and noting that these are based on the internal clocking speed, not the reference oscillator frequency) it may be possible to get a much greater usable frequency range.

There are higher speed versions of the AD9850/51 available that can generate a stable signal at 1GHz or more. But these chips are much more difficult to work with - but who knows



Additional Material

The pictures: The first picture (above) is the final product. Second is of the completed unit before being buttoned up. The third is the ZIF

jig used while experimenting with the PIC program.

It's not practical to insert and remove the PIC chip repeatedly from the pcb without damaging either it, or the socket. The ZIF socket gadget was built to facilitate experi-



menting with the PIC programming.

A ZIP file (the first URL) contain the Gerber files, NC and tool files for the printed circuit board. This file also contains the circuit (in a CircuitMaker file) and a layout in TraxMaker format. The software listing in PDF is at the second URL, and as a text file at the third. Another version of the N2APB software listing contains more notes that may be useful if you decide to modify the software extensively. It's at the last URL.

<http://skirrow.org/Boatanchors/docs/dds.zip>

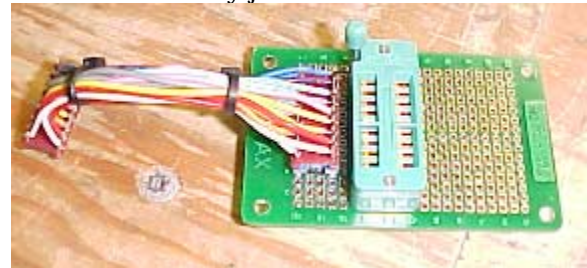
<http://skirrow.org/Boatanchors/docs/pic.pdf>

<http://skirrow.org/Boatanchors/docs/pic.txt>

http://www.njqrp.org/mbrproj/dds_vfo2.txt

Additional Thoughts

After some experimenting, I replaced the optical encoder I had intended to use with a Grayhill encoder that has a push button on the shaft, and 32 increments per revolution rather than the 128 on my junkbox encoder.



It's clear that the LCD is a source of difficulty. LCD operations are slow, and the PIC programming attempts to take this into account. However, each brand of LCD seems to

be somewhat different. The DDS unit I built has two glitches.

First, it is possible to "outrun" the LCD by tuning too fast. The LCD seems to get confused, and can lock up, requiring a power down reset. The DDS chip, however, continues to function and accept instructions from the optical encoder/PIC circuits. In normal use, this rarely is a problem.

Second, occasionally one LCD digit has a nonsense display. Presumably the timing loop in the program isn't quite right. This glitch persists only until the encoder is turned. The display then corrects itself.

I think these are fixable problems, but as the unit is more than adequate for my purposes, I don't intend any further work. I do intend to produce an AD9851 version.

Table 2 - Sample Digit Weight Calculation - 10M Digit

| <i>$\Delta Phase = 650747592.10193$</i> | | | |
|--|----------------------------|----------------------------|------------------------|
| Divide by 16 | Remainder Mult x 16 | Remainder (decimal) | Remainder (hex) |
| 650747592.10193/16 | 0.564 x 16 | 8 | 8 |
| 40671724/16 | .75 x 16 | 12 | C |
| 2541982/16 | .88 x 16 | 14 | E |
| 158873/16 | .5625 x 16 | 9 | 9 |
| 9929/16 | .5625 x 16 | 9 | 9 |
| 620/16 | .75 x 16 | 12 | C |
| 38/16 | .375 x 16 | 6 | 6 |
| 2/16 | .125 x 16 | 2 | 2 |

Digit_weight for 10MHz digit is hex 26C99EC8