Controller for AD9850 DDS Modules

Andy Talbot G4JNT 2012-12-06

Latest Comments. Construction notes and feedback from builders at the end

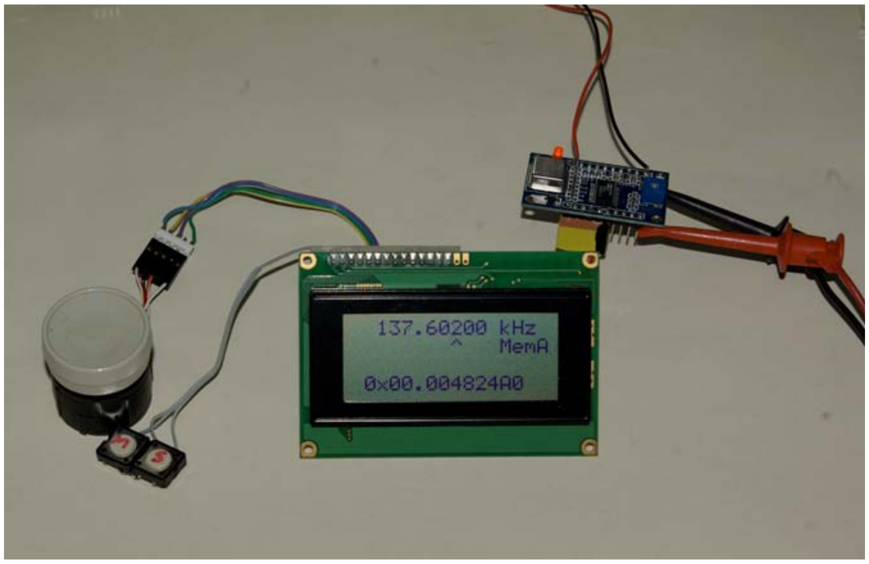
The low cost (£3) Chinese made modules available via Ebay contain an AD9850 DDS chip, 125MHz clock oscillator but no controller for setting the chip via its three wire SPI interface. The modules run from a 5V supply and consume about 120mA.

This unit provides an easy means to set the frequency to any value up to 42MHz by use of a rotary encoder, two pushbuttons and LCD. 16 non-volatile memories are provided allowing any of 16 prestored frequencies to be called up. Each frequency can be adjusted with the rotary encoder and stored back into that memory position if needed.

The circuit diagram is shown below… Communication with the DDS module is by the three wire serial interface, going to the *Data*, *FU-UD* and *WClk* pins. [1] A fourth line also controls chip *Reset*. This latter is not essential, but has been included as it may be useful in the future to have a reset capability. A 5V regulator is provided on the controller. This is only rated at 100mA so shouldn't really be used to power the DDS chip. It does work, but you have been warned!

The software in the PIC maintains a 32 bit frequency word that is incremented or decremented from the rotary encoder. This word represents the desired frequency in units of 0.01Hz, and is independent of the DDS clock. The value being converted to BCD and displayed on the LCD (inserting a decimal point at the appropriate place). The 32 bit word in units of 0.01Hz means the highest frequency that can be stored is about 42.94MHz which is just about the maximum possible this clock frequency.

To generate the 32 bit code to send to the DDS chip, this 32 bit value is multiplied by a constant equal to Step size / Fclock * 2^{64}. Here, is it 0.01Hz / 125MHz * 2^{64} = 1475739526, or in hex notation 0x57F5FF86. This is stored using 6 bytes to allow for lower clock frequencies and / or larger step size The highest 16 bits, or four bytes, of the result form the value sent to the DDS.

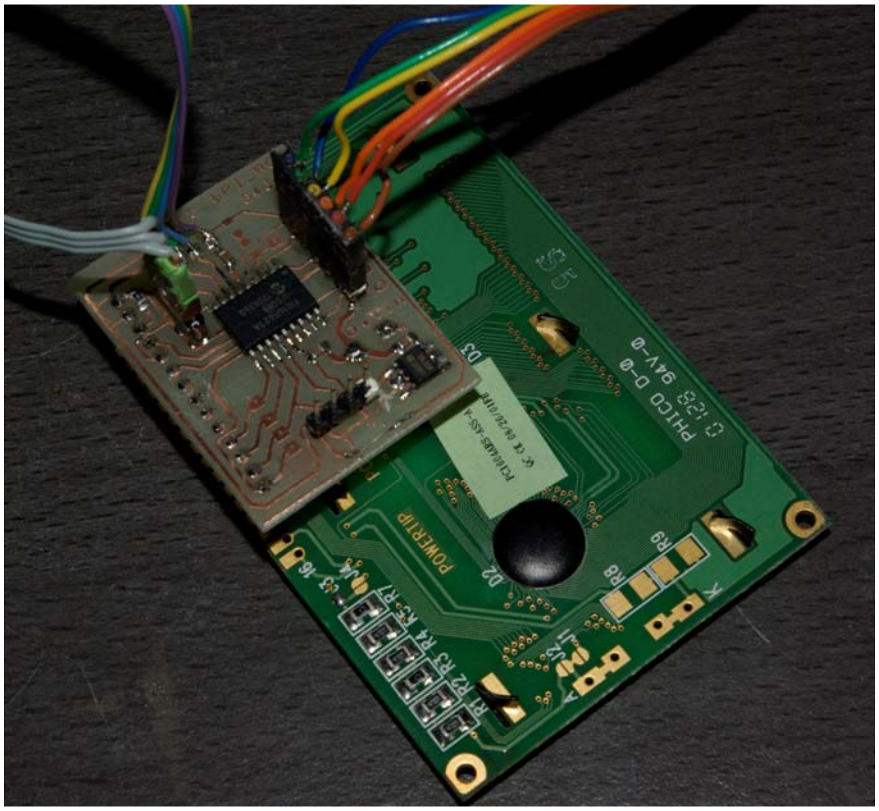


The tuning step, corresponding to each pulse from the rotary encoder, can be cycled though each decimal digit from 100kHz steps down to 0.01Hz by pressing the S pushbutton repeatedly. The digit being changed is shown by the caret on the second line of the display. Holding the S button down for 2 seconds stores the displayed frequency in the current

[1] . This module is also used for control of Fractional-N synthesizers with appropriate PIC firmware. Ignore the part on the diagram referring to LMX2xxx connections.

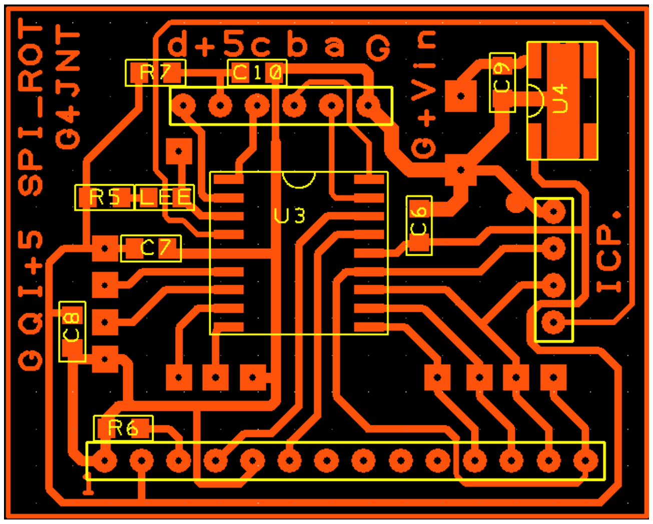
memory.

The 16 memories can be cycled through with the M button. At turn on, the first memory location, MemA is always loaded. Cycle through each location as desired, using the rotary control to move from this value.   Holding S down for two seconds stores the new value in that location.



A minimum step of 0.01 Hz may seem a bit pointless when the fundamental DDS resolution using the supplied clock is 125MHz / 2^{32} = 0.029Hz, but it has been included for when the module is used with a lower clock frequency.   For example, a 10MHz clock allows 0.0023Hz resolution.

PCB



A PCB layout for SOIC packaged 16F628A PIC devices is shown below. The 14 pin header plugs directly onto the LCD modules available from [1], with all other connections made via pads or headers.   Construction should be self explanatory.   The LED serves no useful purpose in this designs and can be left out.   It is a hangover from an earlier project that used this PCB.   (The odd pin connections for the serial interface to the DDS module are also a legacy of backwards compatibility)

PIC Firmware

Source code for programming into the PIC is contained in the file *9850_rot.asm* which includes sufficient comments and notes if you need to modify it to change clock frequency, step size or whatever.   The 6 byte word forming the tuning constant, that needs to be changed for other clock frequencies, is stored in the six lowest EEPROM locations.

Compiled code for programming direct into a 16F628A for use with the 125MHz clock oscillator can be found in *9850_rot.hex*   (The code is just about small enough to fit into a 16F627A device)

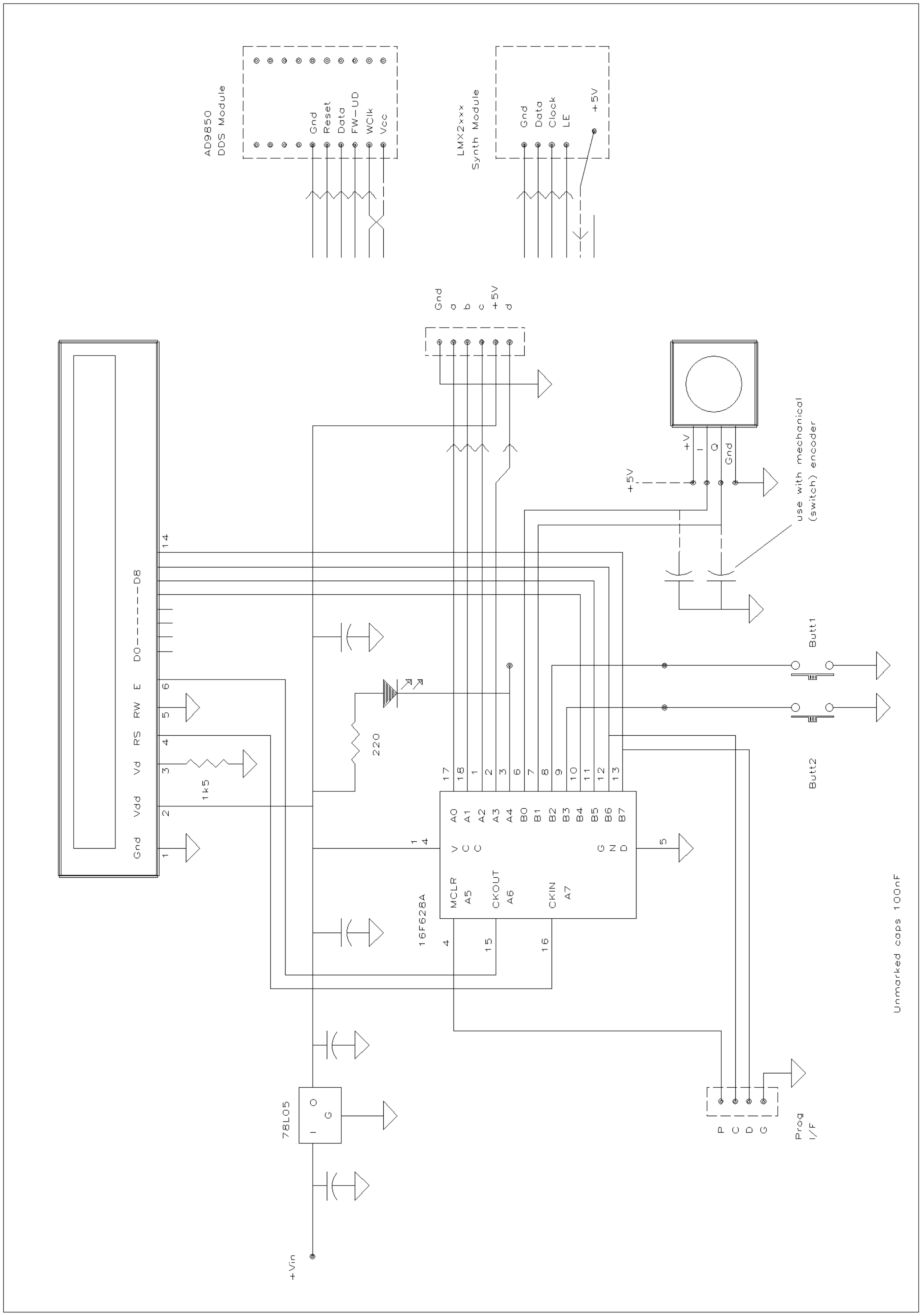
The file *SPI_ROT.pdf* is a 1:1 mirror imaged monochrome plot of the PCB for direct laser printing to acetate or iron-on transfer for homebrew PCBs.

All files can be found at www.g4jnt.com/AD9850-Controller.zip

LCD Modules and PCB availability

Low cost suitable LCD modules at £5 (TBC) each can be obtained from Kevin G3AAF. Contact him at Kevin@avery03.fsnet.co.uk .

The module is so simple that commercially made PCBs are an unnecessary expense.   However, I *may* be able to supply home produced ones. Contact ac.talbot@btinternet.com if you are interested.



Notes, Feedback, Comments

PIC Type

Although a 16F627 PIC is specified in the assembly source code, a 16F627A, 16F628 or 16F628A can be used without change. Most programmers even won't blink an eye at the wrongly specified device. If your particular programming software does whinge, just change the appropriate lines of code and reassemble.

Rotary Encoder

The circuit diagram implies an optical or otherwise powered encoder. A simple mechanical switch type works equally well, ignoring any connection to +5V. There is no need for pull-up resistors, the PIC already has them enabled. However, to avoid contact bounce, two capacitors across the switch contacts will probably be needed. Use a value in the region 10 – 100nF

Two-Line LCD

Many have asked if a 2-Line LCD module can be used insteadof the four line one shown. Those who have tried it reported it worked. The frequency and memory data is still shown. Only the hex register contents are missing.

Startup MemA Issues (modification 2013/05/27)

Several users reported incorrect recall of the Memory A value at switch on, resulting in a random frequency being loaded. This appeared to be particularly troublesome when mechanical encoders were used with debounce capacitors.

The problem appears to be caused by the INTF interrupt being enabled too soon, and corrupting the values loaded from memory. As a quick fix, both INTF and the global interrupt have been moved to the end of the Startup routine which appears to have 99.8% cured the problem. Now, there may be a spurious but minor change of just one digit under the cursor at startup.