All Things Digital

Amateur Radio for the 21st Century

019

Robert C. Mazur, VA3ROM
E: va3rom@gmail.com
W: http://www.va3rom.com

First published in the Jul-Aug 2015 issue of The Canadian Amateur

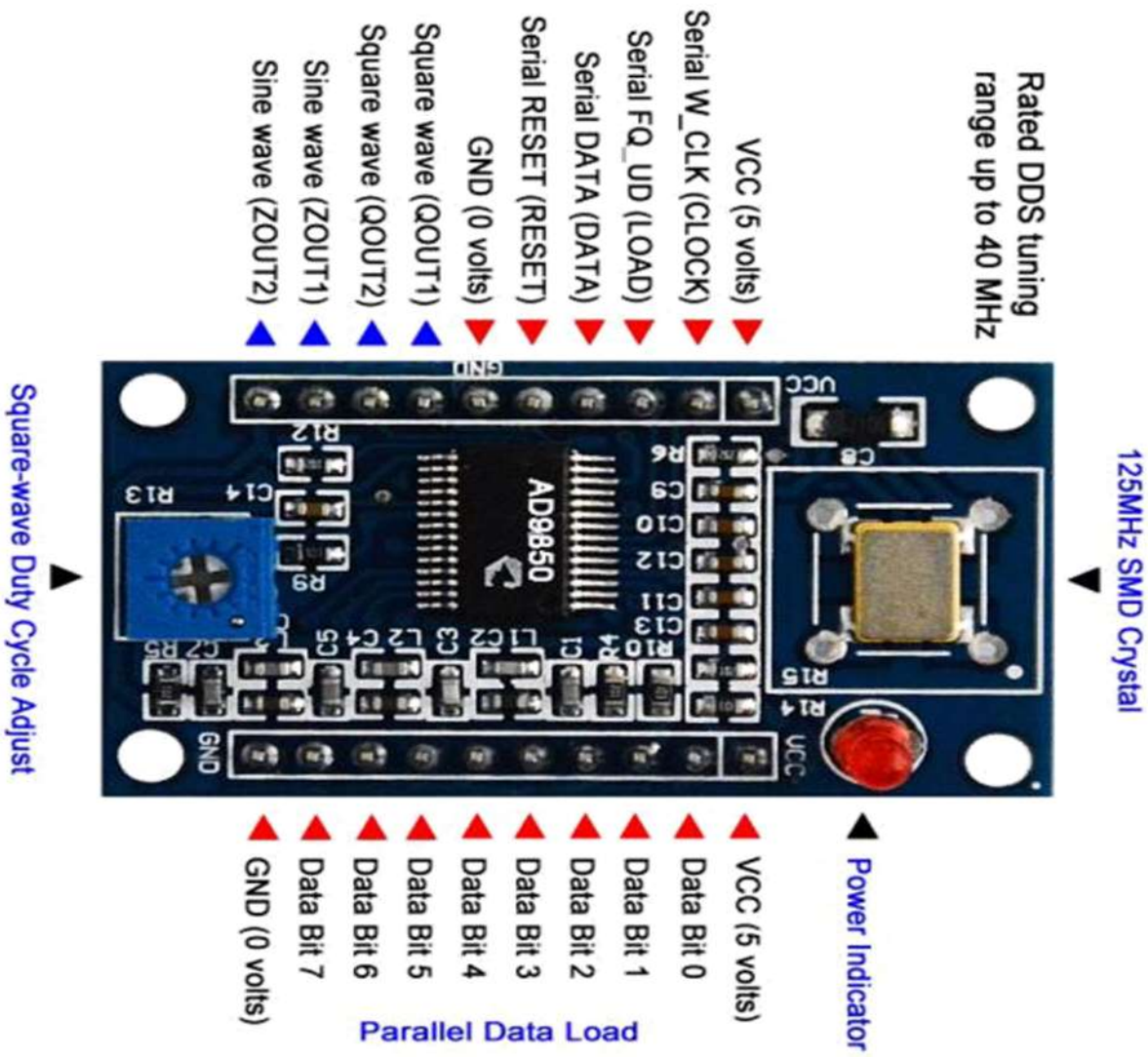
An Arduino DCR-SDR Project: Part 2

INTRODUCTION

Part 1 covered the basic theory and construction of a direct conversion receiver (DCR). Now we'll add an Arduino Uno microcontroller unit (MCU), an AD9850 direct digital synthesis (DDS) module, a liquid crystal display (LCD), and the supporting hardware and software to complete the "gadget". Simplified technical explanations and diagrams are used to make concepts easier to follow. Special thanks to Paul Darlington, M0XPD, and Dr. David Mills, G7UVW, for their invaluable assistance.

DIRECT DIGITAL SYNTHESIS (DDS)

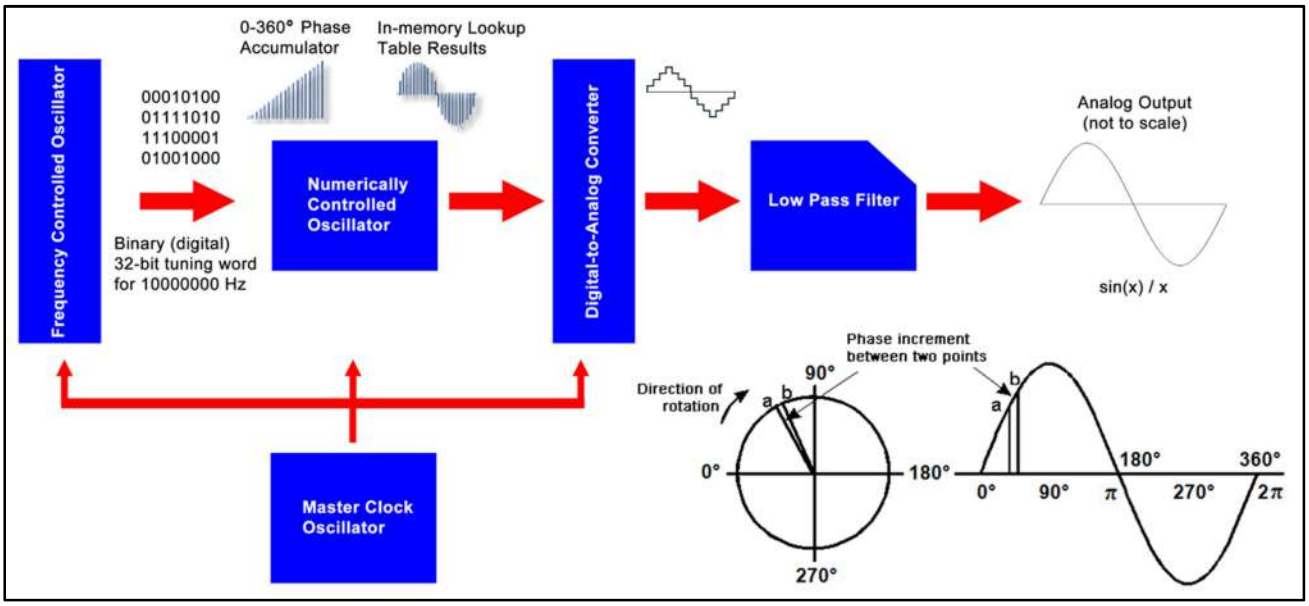
In a nutshell, DDS generates audio or radio frequencies (AF or RF) using software to mathematically manipulate the output of a hardware based master crystal oscillator (MCO). A DDS provides the local oscillator (LO) RF signal for our DCR to demodulate incoming radio signals, which produces both a new RF signal (the product or sum) and an AF signal (the difference). The AF signal is fed into a computing device's soundcard for digital signal processing (DSP) software to decode and display as text, sound, and/or images. Our LO is built with the very popular, inexpensive, and easy to use AD9850 DDS module (image next page); it can generate RF sine/square waves from up to 40 MHz, but above 20 MHz the signal has more harmonics and lower output.



GENERIC DDS STRUCTURE AND OPERATION (image next page)

1. A crystal controlled MCO from which all other frequencies are derived (synthesised) as some fraction thereof. The theoretical maximum or Nyquist frequency is one-half the MCO frequency (62.5 MHz, in this case).
2. A frequency control register (FCR) to mathematically convert the requested frequency from decimal to a binary (digital) 32-bit “tuning word”. Actually, 40-bits are used with 8 reserved for signal phase, power down control and [data] loading method (serial/1-bit at a time, or parallel/8-bits at a time).
3. A numerically controlled oscillator (NCO) consisting of a phase accumulator (adder) with an in-memory lookup sine table (14-bit or 16384 values).
4. A digital-to-analog converter (DAC) to convert digital output into a “real-world” analog sinusoidal wave (AD9850 uses 10-bit DAC).

5. A low-pass filter (LPF) to smooth the analog signal and remove harmonics (to a certain point) but with a big “but” because secondary/unwanted signals called “aliases” (“aliasing”) increase with increasing frequency, and are especially noticeable above 20 MHz (for the AD9850).

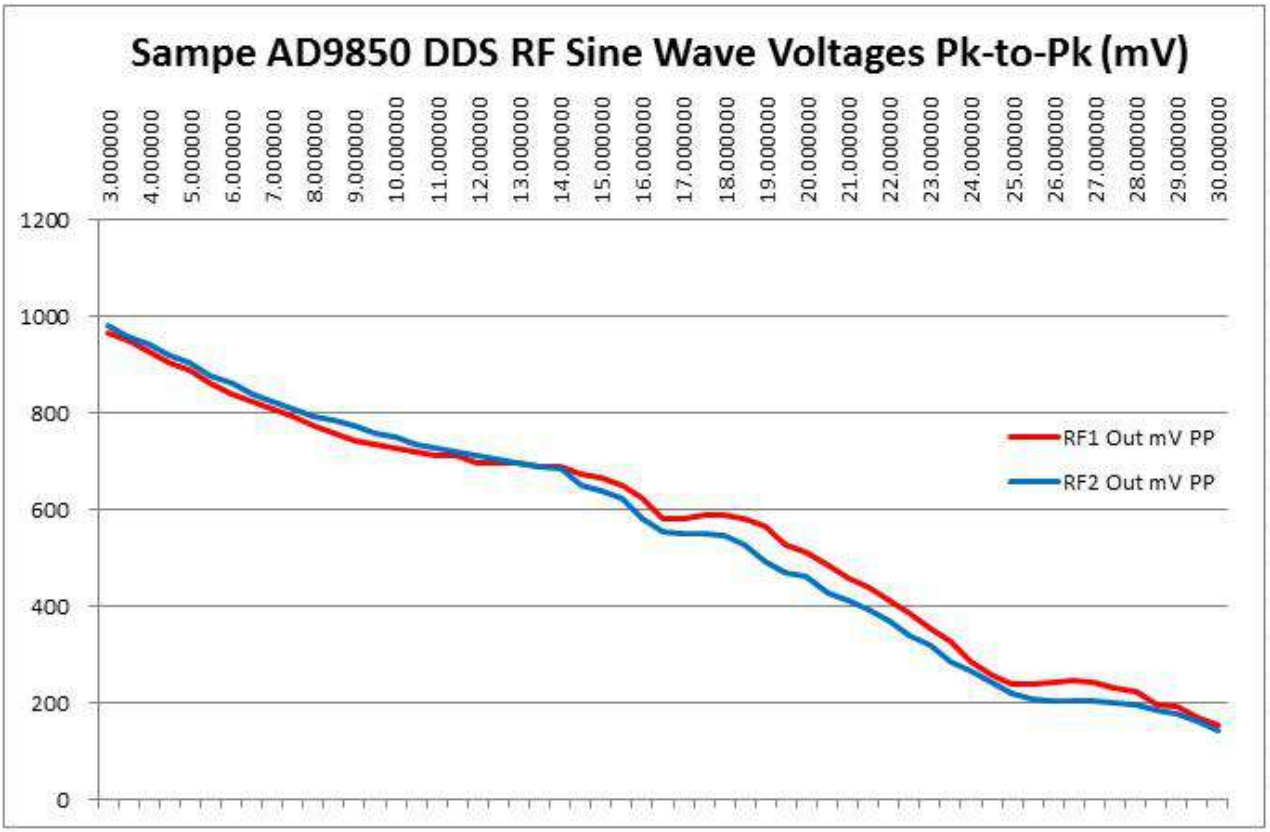


DDS CAVEATS

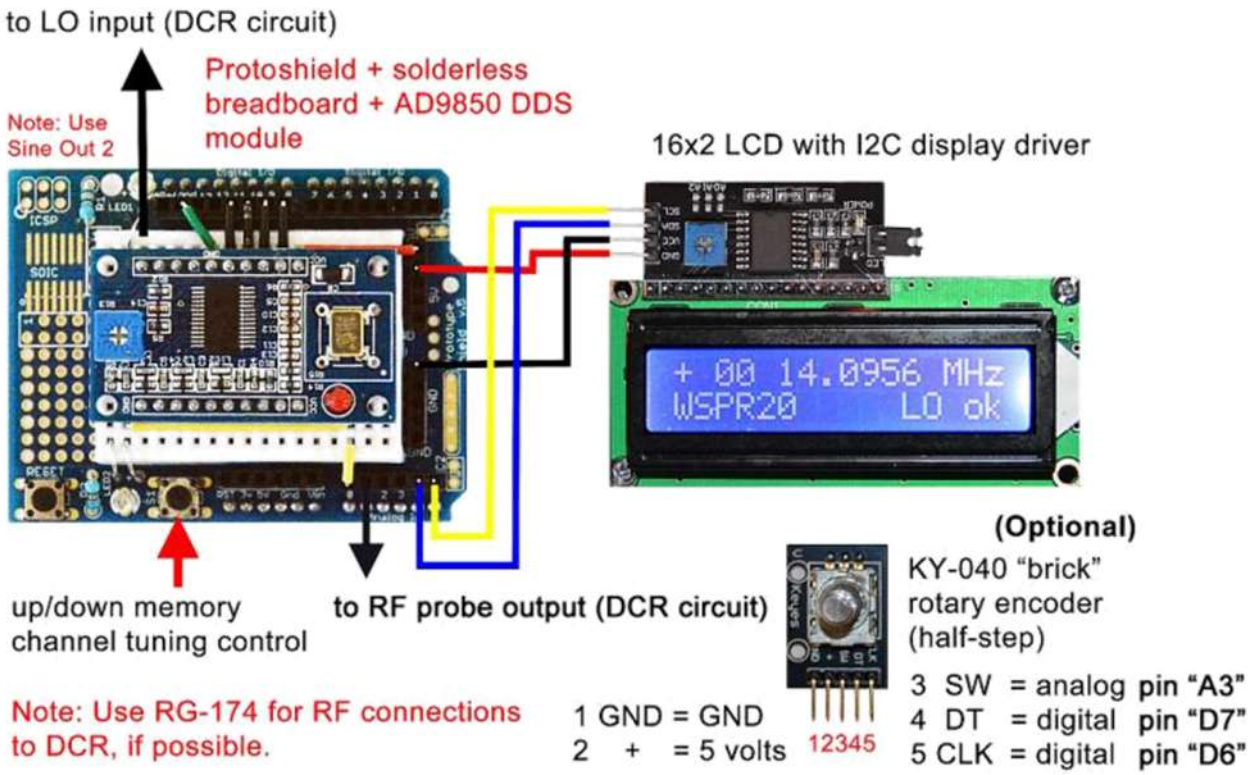
Note: The exact frequency we want is usually not the one generated by a DDS module until it has been calibrated against a reference signal at a constant voltage and temperature. The MCO crystal will have frequency error measured in parts-per-million (ppm) for which you must determine and correct.

A stable, well-filtered, 9-12 volt DC power source (or battery) using short (< 1 metre) leads is preferred because the MCO is supply voltage (Vcc) sensitive; even a +/- 0.1 volt change will shift frequencies up/down. Except for programming your MCU, don’t use USB or AC/DC “wall-warts” to power ancillary connected RF devices because these produce RF “hash” or have ripple/hum. The USB volt output is also affected by the length/quality of the connecting cable so it’s rarely at 5 volts (varies from 4.5 to 5.5 volts DC).

Many inexpensive DDS modules (like the AD9850) don't have a temperature controlled crystal oscillator (TCXO) and therefore the MCO is very sensitive to ambient air temperature changes, and therefore requires a housing of some kind to reduce frequency shifts. If possible, a heatsink should be adhered to the MCO crystal (can be metal or plastic). Frequency errors are also caused by the "cut" of the MCO's crystal and most don't resonant exactly at the frequency stamped on their cases.

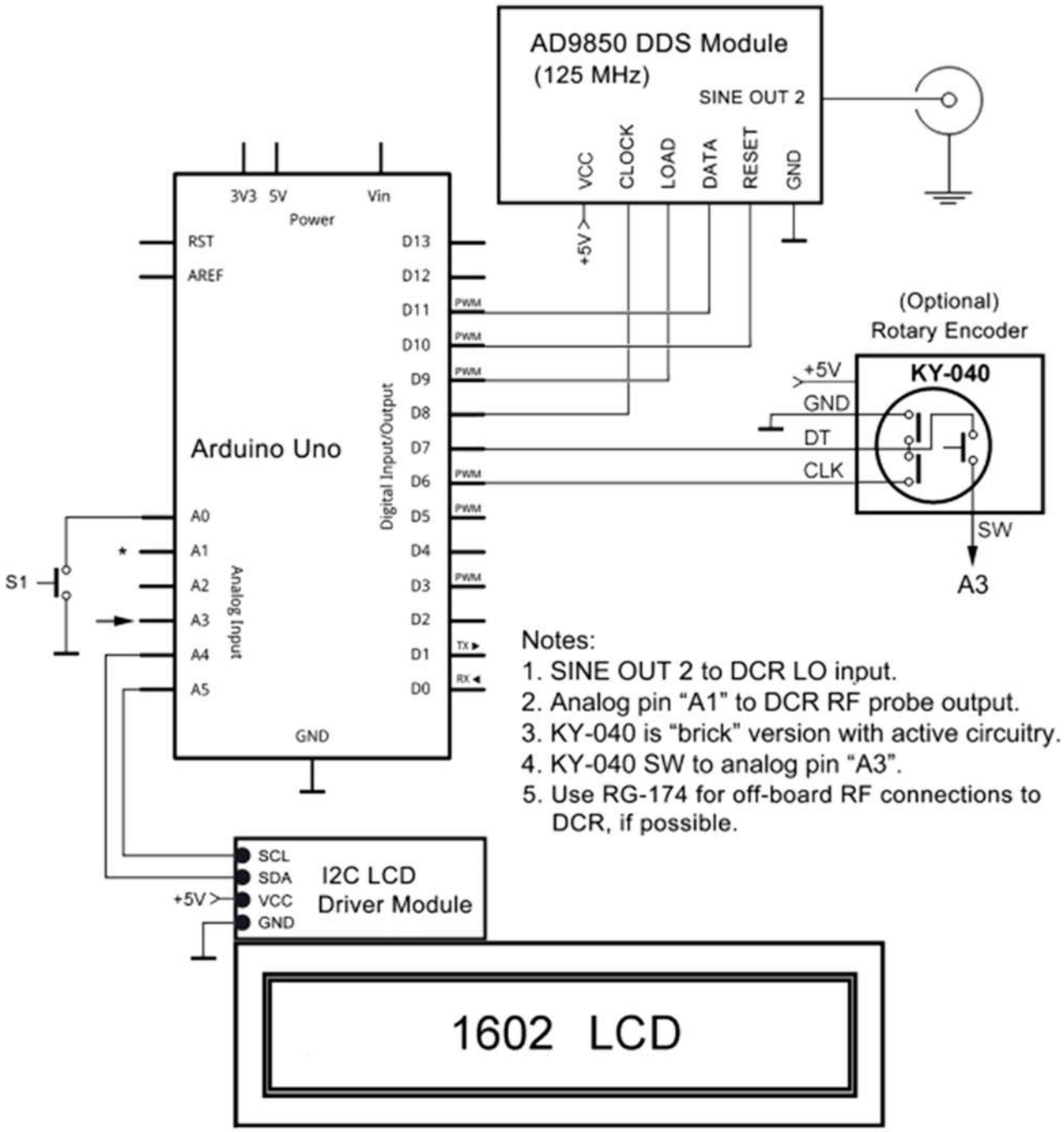


The AD9850's sine wave RF output voltages aren't constant across the tuning range (above image) because of the sine cardinal (SINC) function or SIN(X)/X. This "plagues" most DDS devices because they don't have automatic level control (ALC) circuitry. For the AD9850, it's just over 1 volt (peak-to-peak) at the very low end, and decreases as frequency increases. This is the reason for our DCR's LO level adjust potentiometer (VR1) because the SA612 Gilbert cell mixer requires a drive RF voltage between 200-300 millivolts (peak-to-peak) so it needs to be readjusted when changing the LO frequency.

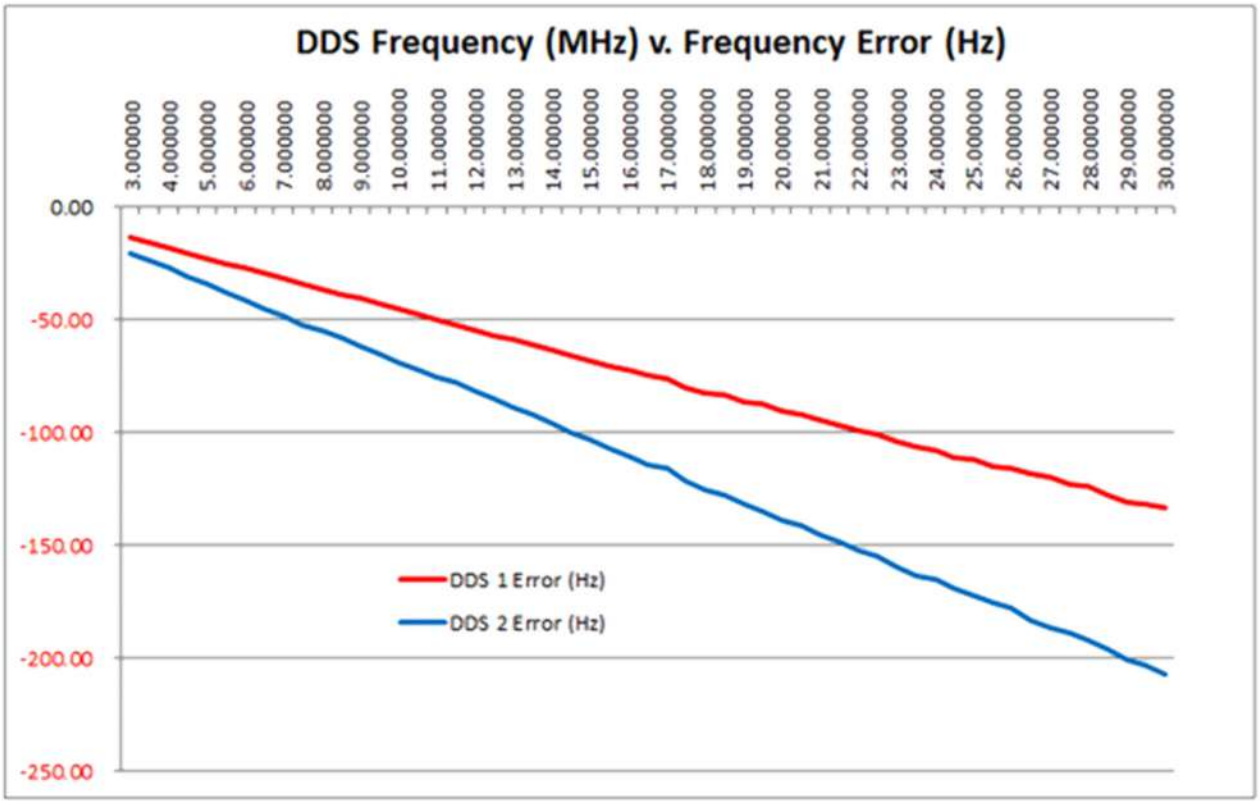


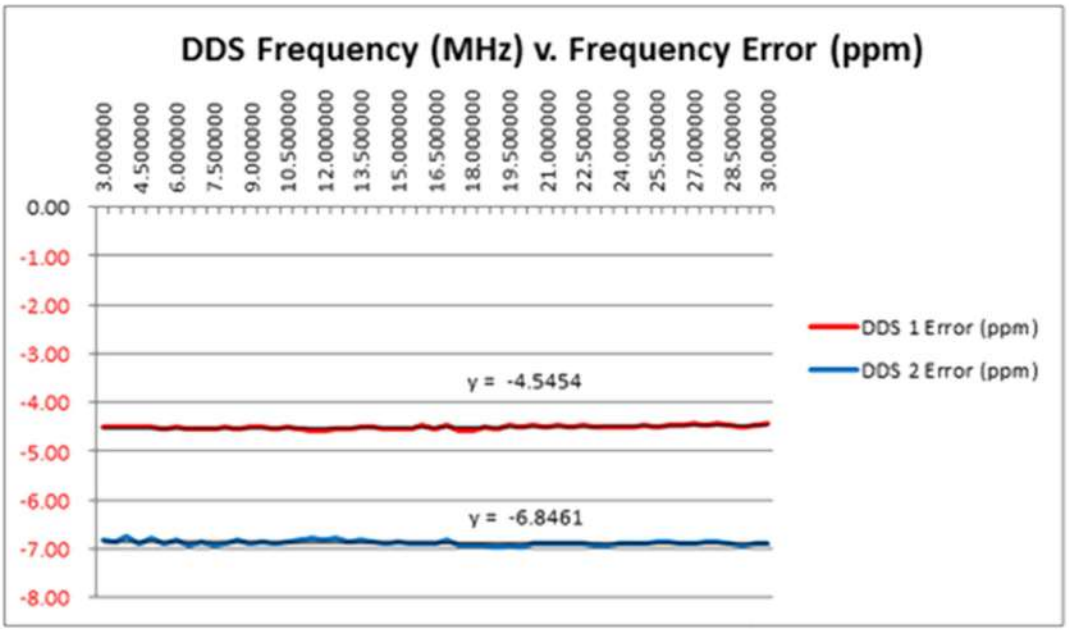
THE VARIABLE FREQUENCY OSCILLATOR (VFO)

You can use almost any MCU with the AD9850, but the Arduino has a lot of public or "open-source" support code making it a breeze to use—even for non-programmers. By combining an Arduino Uno (in this case), AD9850 DDS module, LCD, and some program code, you have created a very handy, stand-alone RF variable VFO gadget to drive external circuits like a receiver/transmitter as an LO, or test/calibrate or troubleshoot other electronics, etc., so it's not just limited to this article's application (above image and schematic next page).



Because I'm only interested in radio data modes which are fixed frequency, a rotary encoder (the digital equivalent of an analog rotary switch) wasn't included in my original circuit. Up/down "tuning" is done via a single pushbutton ("S1") mounted on a breadboard shield. But after finding the excellent code and tutorial written by Ben Buxton, I added a KY-040 "brick" rotary encoder for those who prefer to use them.

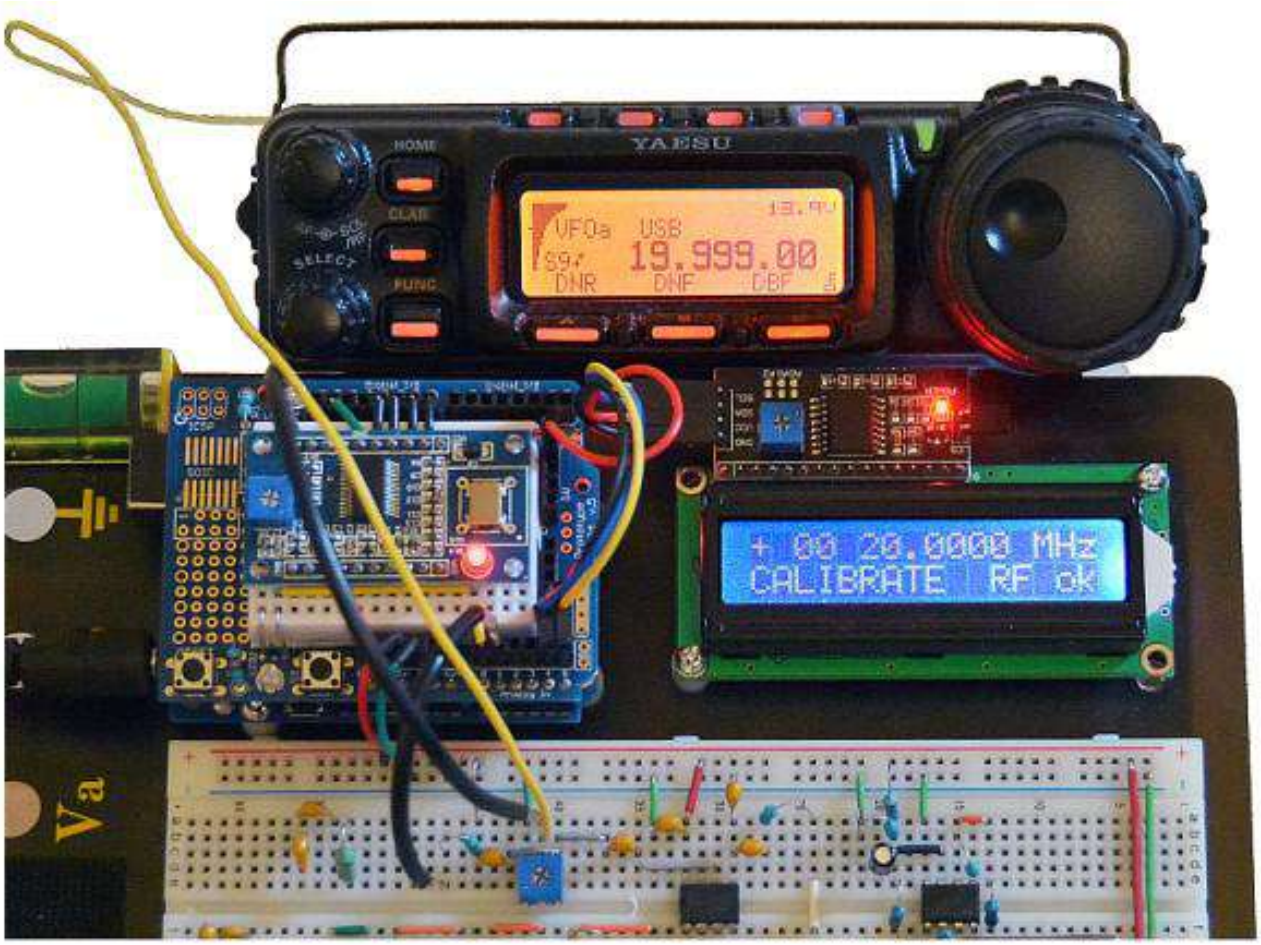




CALIBRATION, CALIBRATION, CALIBRATION

The above image is a composite of two graphs; the top one depicts frequency errors for two apparently “identical” AD9850 DDS modules, but it’s very clear that their individual frequency errors aren’t identical and you just can’t add (or subtract) some fixed frequency to correct for this if you require multi-frequency operation.

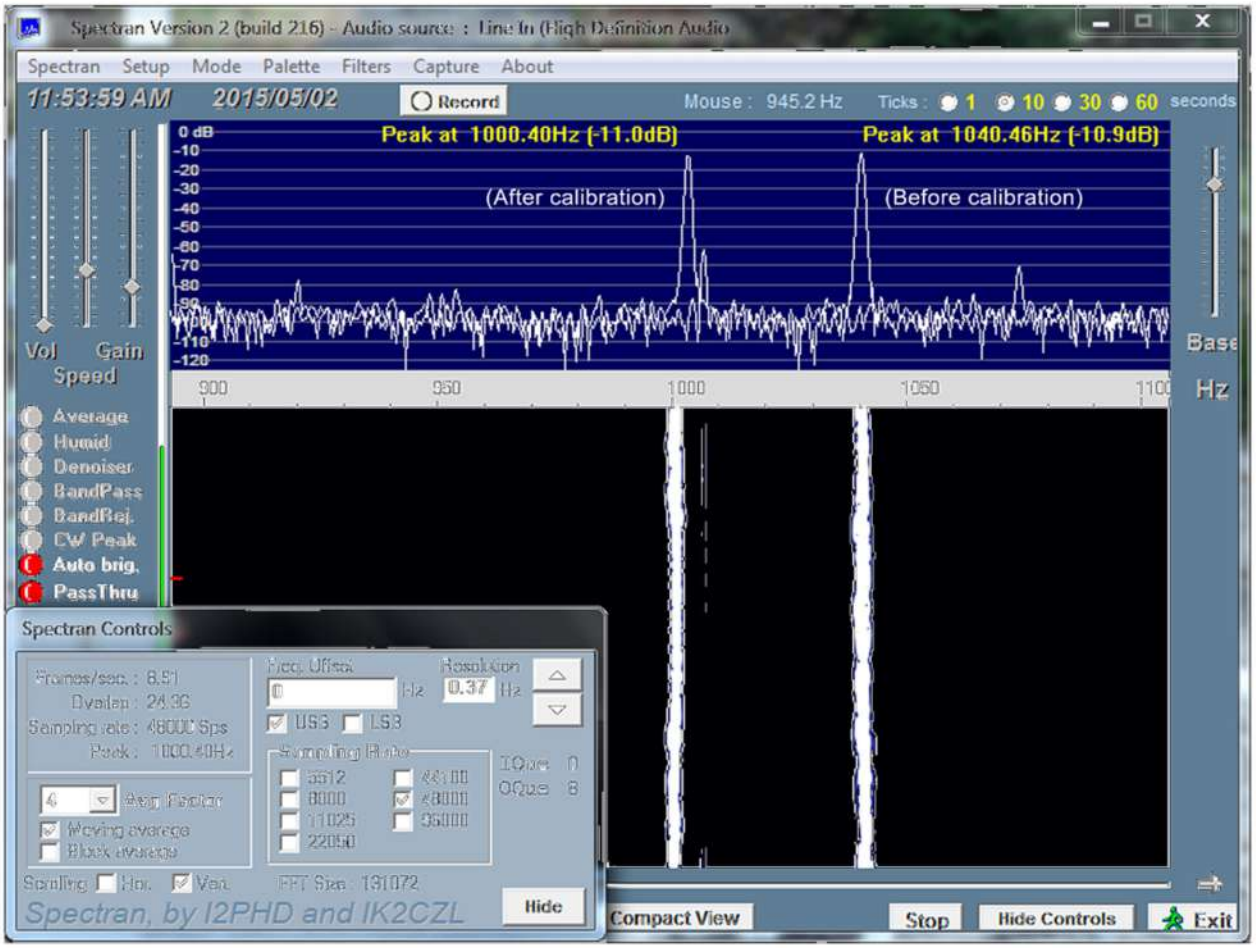
However, both plots have a constant slope (second graph) we can calculate, and this value measured in ppm is constant across the entire DDS tuning range. Now, testing individual DDS modules and creating graphs is very time consuming and tedious, but there is another calibration method using audio DSP programs like Spectran (also Argo or Spectrum Laboratory) plus a reference radio.



Spectran Software DDS Calibration Method 1 (image above):

1. Disconnect outside antenna from the reference receiver or transceiver.
2. Switch it to USB receive mode (no RIT or IF shift, etc.) and tune to exactly 19.999.00 on the dial (1000 Hz lower than the DDS calibration frequency).
3. Power up the programmed Arduino (DCR-SDR code version) and select memory channel 0 “CALIBRATE” 20 MHz using pushbutton “S1”, or manually tune to 20 MHz (DDS VFO rotary encoder version).

4. Couple the DDS RF sine wave output 2 (it uses the onboard DDS 70 MHz low pass filter [LPF]) with a length of wire to your radio's antenna input (centre of antenna jack). DO NOT USE the DDS square wave outputs!
5. Connect an audio cable from the radio's speaker or AF line output to your computer's soundcard line or microphone input.
6. Start Spectran and slide the horizontal AF frequency display (grab and slide with mouse) and centre it on 1000 Hz. Adjust the radio volume and soundcard mixer controls so you can see an audio signal peaking somewhere in the Spectran spectrum display near 1000 Hz with a white vertical line dropping down the waterfall (image below).



7. Let everything warm up and reach electronic thermal equilibrium (at least 30 minutes) before making measurements/calculations. Keep the room temperature constant (I used 20 °C).

8. The audio signal's frequency and trace will vary slightly, so take a reading every minute and average ten. Your soundcard has a crystal controlled MCO (or two) as does your computer and receiver and their collective/cumulative frequency errors must be taken into account.
9. The difference between the two RF frequencies (20 MHz and 19.999 MHz) produces a fixed 1000 Hz audio beat note if the DDS generated frequency is spot on. Calculate the parts-per-million error by subtracting the average audio frequency from 1000 and dividing the result by 20.

e.g.: My DDS Spectran audio frequency average (10 readings) is 1040.34 Hz:

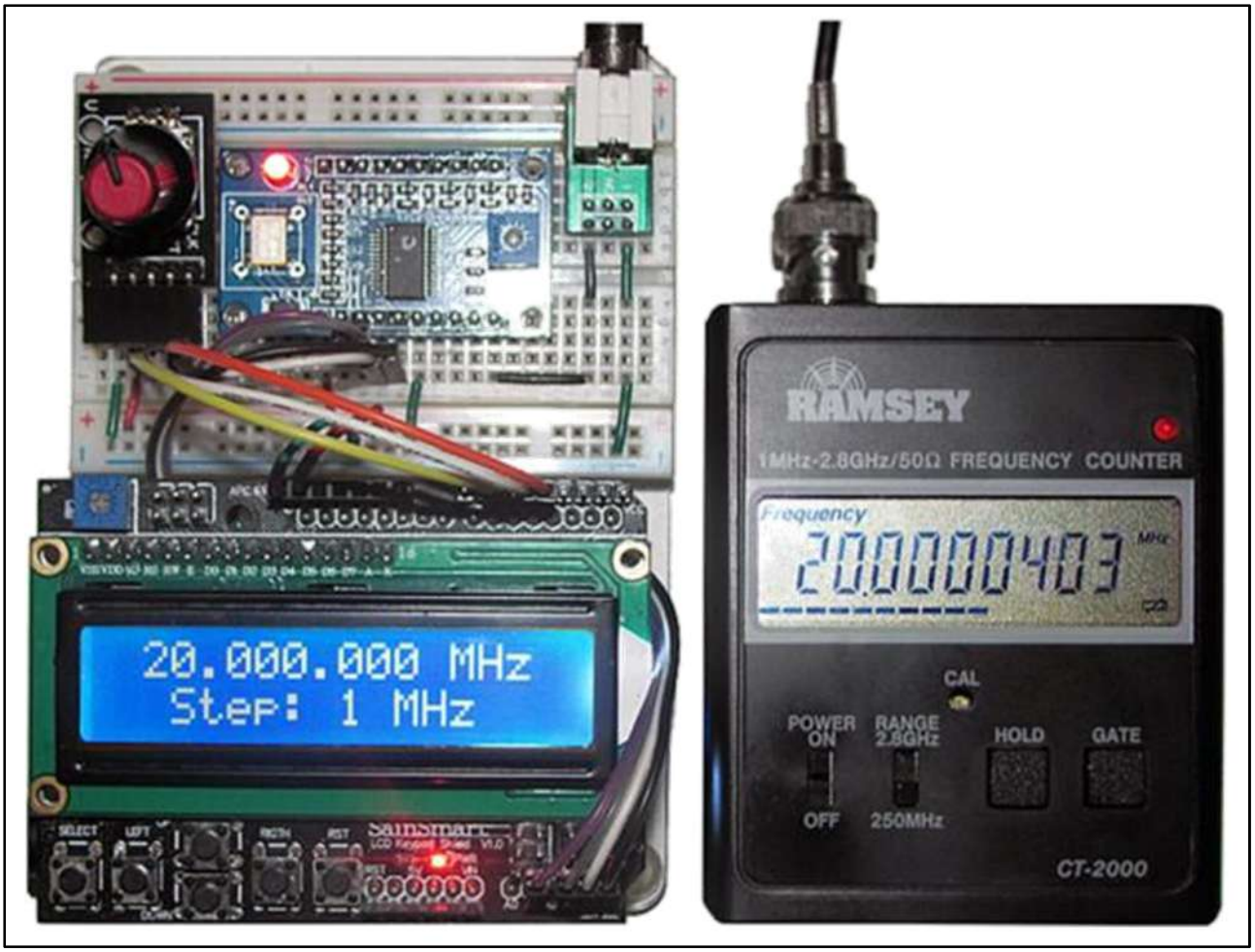
(1000 – 1040.34) / 20 = -40.34 / 20 = -2.017 ppm

This specific module's ppm error correction is negative because it generates DDS frequencies too high, but others can be too low (positive ppm error correction). Assign this to the DDS_CORR value (in both Arduino programs). We can now use the software to mathematically "pull" this specific DDS module's MCO crystal by the proper amount to correct for frequency errors across its entire tuning range.

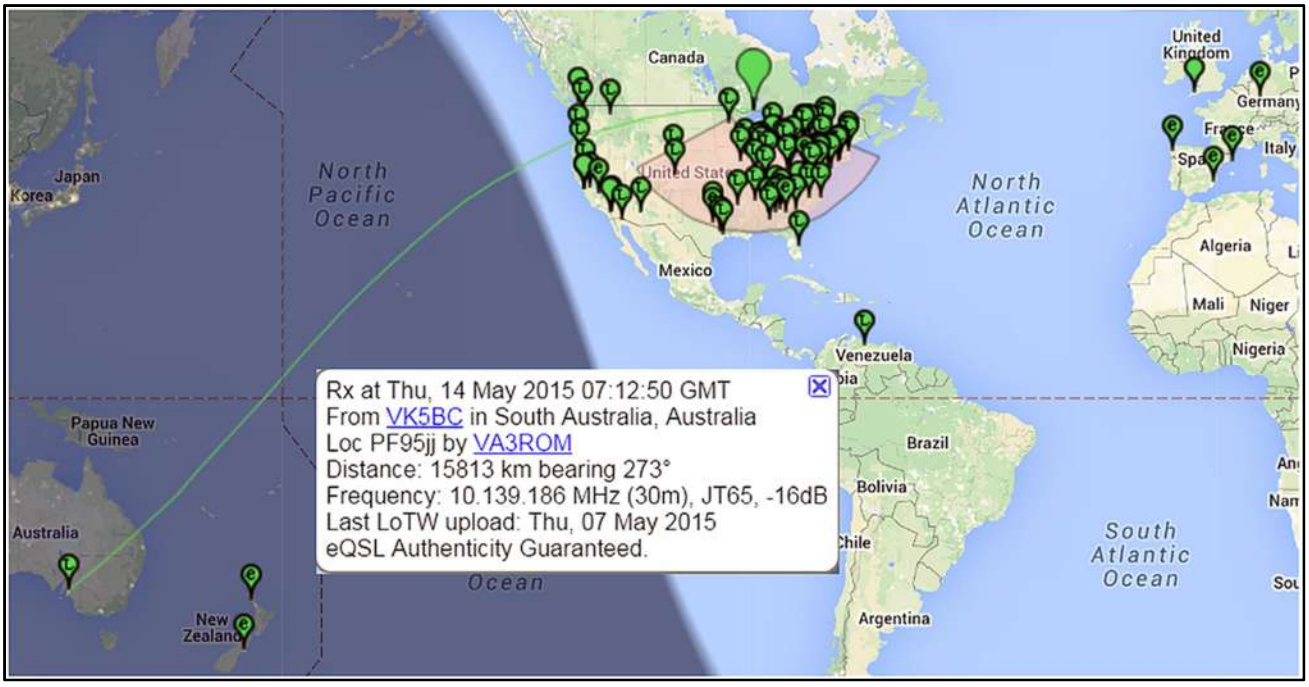
Faster DDS Calibration Method 2 (image next page):
If you want to simplify and speed up things, use an RF frequency counter. While it's not as educational or demonstrative as the first method, it's a necessity when you have to calibrate umpteen DDS modules (stick a label on each with their ppm correction). In this case, I'm subtracting the RF DDS and frequency counter frequencies (both in MHz) to determine their difference (in Hz) then dividing by 20:

e.g.: (20 – 20.0000403) / 20 = -40.3 / 20 = -2.015 ppm

While method one is slightly more accurate, method two is faster and good enough until we house the circuitry and must recalibrate for the increased ambient air temperature inside the case (from heat generated by the electronics).



The above image is of an alternate MCU version eliminating the DCR and just building and RF generator circuit using an LCD shield, half-size solderless breadboard, all mounted on a Plexiglas mounting base designed for the Uno.



WHAT TO DO WITH DATA MODES DATA?

The DCR_SDR program includes a sample list of some popular data mode frequencies you can try using free DSP decoding software such as WSPR, Fldigi, MMSSTV, or MultiPSK. There're many Internet webservers to where you can upload your received data and share with the world: WSPRnet, APRS-IS, World SSTV Cams, Reverse Beacon and PSKReporter (image above). These servers are accessible 24/7 (to anyone) to retrieve data for research, analysis, equipment testing, mode comparisons, and/or propagation experiments, etc.

MY FINAL

Well, that's it—for now. If you don't care to build this gadget, just connect your radio audio output to your computer's soundcard, and go from there. The next four columns look at an analog data mode called slow-scan television (SSTV) used by Amateurs and professionals alike.—73

REFERENCES AND RESOURCES

Audio DSP Software
http://tinyurl.com/nfakd
http://www.weaksignals.com
http://tinyurl.com/33co2h

Data Modes Networks
http://wsprnet.org/drupal
http://aprs-is.net
http://www.worldsstv.com
http://www.reversebeacon.net
https://pskreporter.info

Data Modes Software
http://tinyurl.com/2wgcp2f
https://sourceforge.net/projects/fldigi/files/
http://tinyurl.com/6vblqh
http://hamsoft.ca

David Mills, G7UVW
http://tinyurl.com/ou9yosq

Paul Darlington, M0XPD
http://m0xpd.blogspot.ca

Understanding DDS
http://tinyurl.com/ojkje68

VA3ROM: All Things Digital
http://tinyurl.com/og2acxq