Building the K6BEZ Antenna Analyzer

Lars Anderson, KN5UTE

Jan 10, 2015

This article describes some design decisions, modifications, and programming changes made during my implementation of the K6BEZ Analyzer by Beric Dunn[1]. However, this project can be completed and used without any programming experience and with only a basic understanding of electronics.

Background

I am a newly licensed ham, but had some decades-old exposure to the hobby, and my career involved electronics and programming. I know already that I want to work HF digital modes across as much of the spectrum as I can, with a single antenna. Within the constraints of my 40 ft x100 ft zero lot line home, I installed a 54-foot "balanced doublet" antenna on my roof, based on an ARRL article[2] I found. This antenna can be efficiently used over several bands, but it requires a tuner to optimize the match at any given frequency. I bought an MFJ 941E manual tuner, but in order to optimize the SWR of my antenna I found that I needed to be transmitting a handful of watts to make the needles move visibly. I became concerned about the amount of interference I was spewing around every time I tuned, and decided that I needed a way to determine the optimal tuning of my antenna at any given frequency that involved very low power so that I wouldn't become a local interference nuisance.

I recently enjoyed building a simple project using the Arduino Uno controller board (Fig 1), so I searched the internet for a suitable SWR meter design that included an Arduino. The K6BEZ project seemed to fulfill my needs based on the low parts count, relatively low cost, no SMDs, and essential functionality.

Fig 1

The design uses AA143 germanium diodes for detection, and these are not common. I considered substitution of other germanium diodes 1N34A or 1N60, but ended up mail-ordering the AA143 diodes because I was not comfortable in my analog part substitution prowess.

I found almost all of the parts locally at Tanner Electronics[3]. I ordered the Arduino Uno, Prototype Shield, AD9850 DDS module, and the AA143 diodes from eBay, and the MC6002 op-amp from DigiKey[4]. I used a plastic enclosure, but if I were to do it again, I would use a metal enclosure for RFI shielding.

---

[1] K6BEZ Antenna Analyzer
http://www.hamstack.com/project_antenna_analyzer.html

[2] Balanced Doublet Articles
http://www.arrl.org/random-length-multiband-dipoles
http://www.sgcworld.com/Publications/Downloads/ClassicMultiband.pdf

[3] Tanner Electronics http://www.tannerelectronics.com/

[4] DigiKey http://www.digikey.com/

Construction

Once all the parts were in hand, I assembled the circuit on a solderless prototyping breadboard in order to both test functionality and to experiment with efficient layout of the discrete components for permanent fabrication.

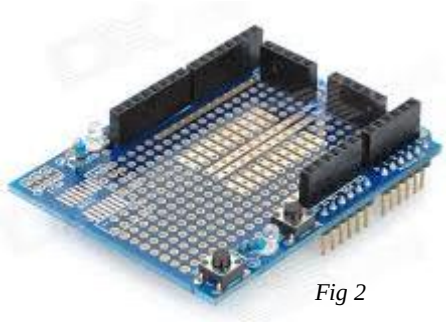


Fig 2

I had originally intended to use the smaller Arduino Nano board, but during the breadboarding exercise, I decided that a plug-in mezzanine "prototyping shield" (Fig 2) for the Arduino Uno would be a good way to mount the discrete components and give direct access to power, ground, and the two necessary Arduino analog input pins.

It took me a few hours to come up with a layout that worked well with the Prototype Shield that I had procured for the Uno, but once the layout was decided, actually soldering the parts down only took me a little over an hour. I used a socket for the op amp chip, and a couple of header pins for connection to the coax connector, and for connection to the DDS Module sine wave output.

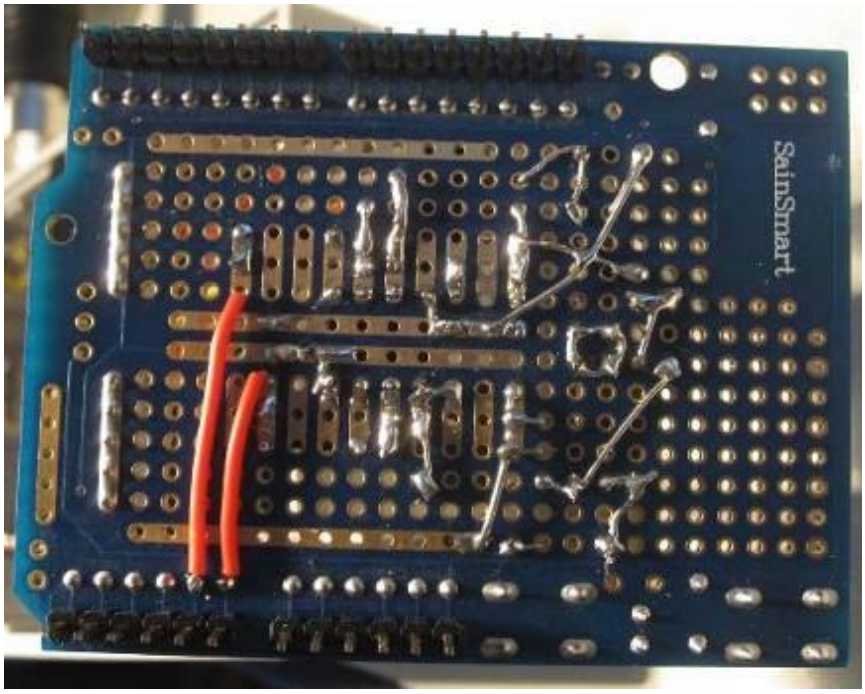


Fig 3 Proto board wiring side

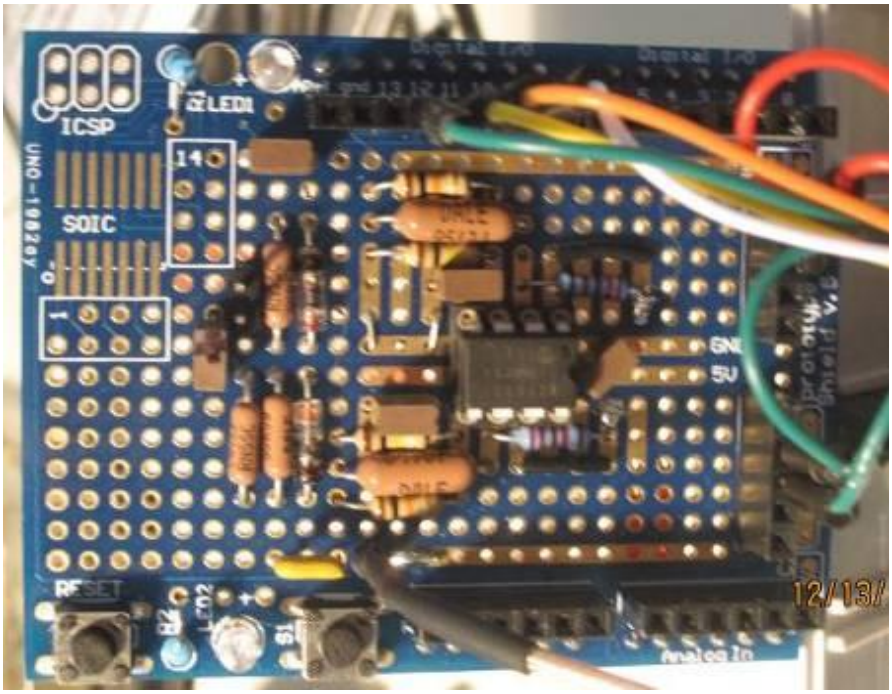


Fig 4 Proto board component side

I decided to use 20cm male-to-male "Dupont" jumper wires to connect the DDS module (Fig 5) to the Arduino Uno shield, rather than using soldered connections. I plugged the DDS module into a very small solderless breadboard (Fig 6) that was attached to the bottom of the project box at one end with double-sided tape. Four data lines, and a couple of Vcc and ground wires (each) complete the set. I also made a tiny coax jumper for bringing the DDS module sine wave output over to the Prototype Shield. I doubt if coax was necessary at 30MHz but it can't hurt. I grounded the shield of the coax only at the DDS module end. The coax cable was built by soldering two Dupont male pins to center and shield of one end of the cable (DDS Module end), and one female Dupont socket to the center conductor of the Prototype Shield end of the cable, leaving the shield unattached at that end.

Fig 5 DDS Module

Fig 6 Solderless breadboard

Physical assembly involved cutting a hole for the USB connector to protrude through the project box, and two holes to affix the Arduino Uno to the bottom of the box (Fig 7). I decided to use a single tie wrap looped through these two holes and matching holes on the Arduino Uno for simplicity and ease of disassembly. For a more permanent assembly, screws and nuts would be preferable.

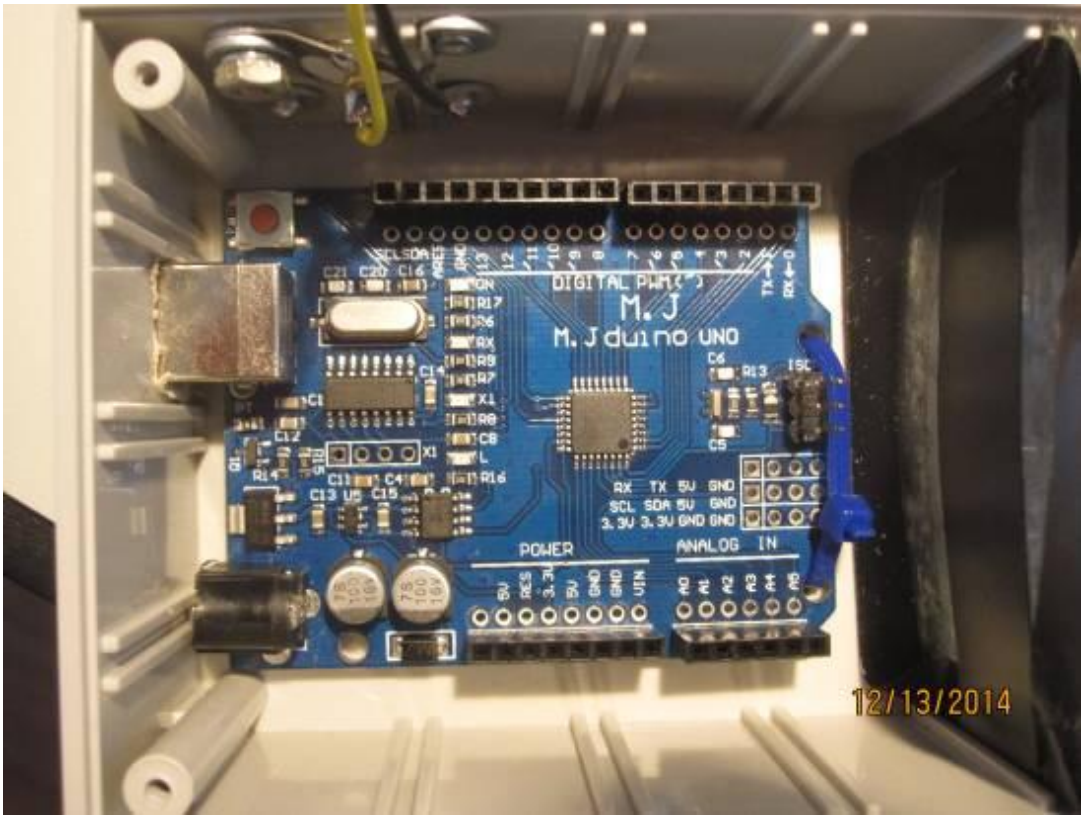


Fig 7 Arduino Uno mounted

The large hole for the SO-239 connector was drilled using a step drill bit which works extremely well. Four screw holes surrounding the large hole were also drilled, but I used pop rivets in three of these holes just for the heck of it. In the fourth hole I used a standard hardware stackup: screw, flat washer, split washer, nut, and also included a soldered ground lug (Fig 8). When completed there is about a 2-inch twisted pair pigtail soldered to the center pin and ground lug of the SO-239, and at the other end of this cable is a 2-pin female header block for connection to the Prototype Shield.

Fig 8 SO-239 detail

The final assembly steps were:

1. tape small solderless breadboard to bottom of prototype box
2. plug DDS module into solderless breadboard
3. mount Arduino Uno to bottom of the other end of the box with a single small tie wrap, leaving the USB connector accessible via the hole we cut previously
4. mount the mezzanine prototype shield onto the Arduino Uno simply with pin/socket headers
5. install the data and power connections between mezzanine board and DDS module
6. install the coax signal jumper between the DDS module and the mezzanine board
7. connect the SO-239 pigtail to the mezzanine board.

Fig 9 Completed Assembly

Programming the Arduino[5]

Arduino Uno is flashed through the USB cable from the free Arduino development environment downloaded for the host platform of your choice. The source code is a standard text file with an *.ino* extension. I used the code that Beric Dunn had provided, but it was occasionally reporting negative VSWRs. I discovered this was due to a typecast error and the correction was to:

Change FROM

*Serial.println(int(VSWR*1000));*

Change TO:

*Serial.println(int32_t(VSWR*1000));*

This fixes the overflow which was causing negative VSWR to be reported on the serial bus.

Another issue has something to do with the DDS module. The VSWR of the first data point pulled by the graphing program was almost always 1, (i.e. 1000 on the serial bus) on the first sweep after power up. The VSWR of the first data point on the second and subsequent sweeps was fine. I band-aided this by inserting one line just outside the *Perform_sweep()* function, to command an initial tone to be generated before the actual sweep loop. I used 10MHz.

Change FROM:

```
//Start Loop
for( int i=0;i<num_steps;i++){
```

Change TO:

```
//Start Loop
SetDDSFreq(10000000);
 for( int i=0;i<num_steps;i++){
```

The PC Program (GUI)

For several reasons, I did not care for the GUI code provided by Beric Dunn, but it was perfectly serviceable and can be used. You should know the code does not support Linux; only Windows versions newer than XP. Aesthetically I didn't like the layout or the font size choices, and the code was written in Visual Basic which I haven't used before.

Through the K6BEZ Antenna Analyzer Yahoo Group blog[6], I contacted Glenn Snedden, VK3YY, who had written a GUI using the *Processing Development Environment* (PDE[7]) that would run under XP. Glenn graciously shared his source code[8].

It turns out that the Arduino development environment is written in PDE so there is no additional learning curve beyond learning the rudiments of the Arduino environment necessary for downloading the controller code mentioned above.

---

[5] Arduino http://arduino.cc/en/Main/Software

[6] Yahoo Group https://groups.yahoo.com/neo/groups/k6bez_projects/info

[7] Processing (PDE) https://processing.org/download/

[8] VK3YY Antenna Analyzer https://vk3yy.wordpress.com/2014/09/29/antenna-analyser-project/

The code that I got from Glenn was quick-and-dirty. It uses an open source library package for PDE called *Grafica* to graph the frequency and VSWR data returned from the controller. It required the user to "hardwire" the USB comm port number, and has some other limitations.

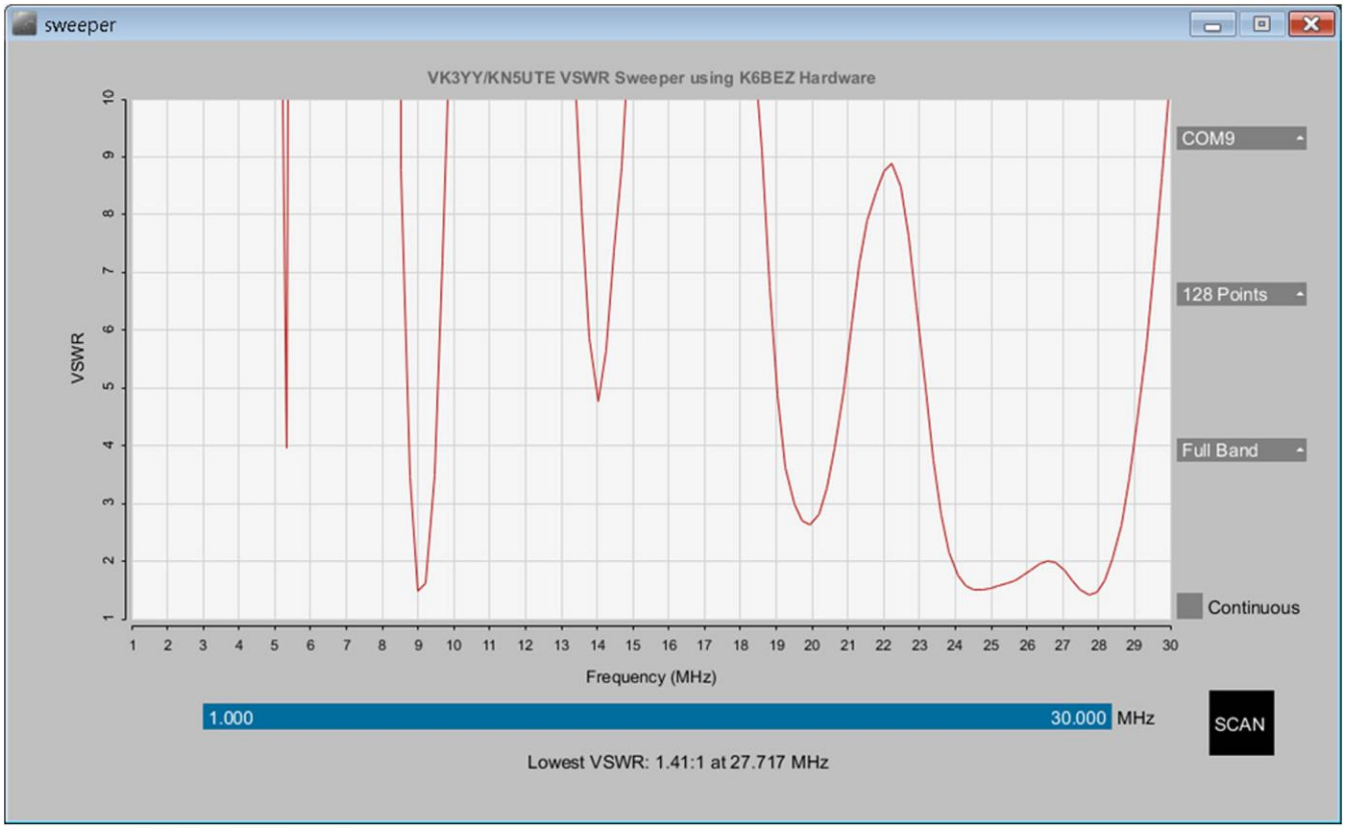
In addition to not being able to select or automatically detect the sweeper USB port, the quick-and-dirty code also had the number of points hardwired at 100, did not do continuous sweeping, and had just 3 "canned" sweep selections: 1-10MHz, 10-20MHz, and 20-30MHz.

I edited the code with the aim of just adding the USB Port selection and Continuous Sweep mode, but ended up doing a major rewrite to the sweeper code. This was made possible by another open source library for PDE called *controlP5*. There are a number of examples online for the supported widgets that *controlP5* supplies. I used the "dropbox" widget for allowing the operator to select the Comm Port, Number of Points, and Canned Sweep Limits. I used the "checkbox" widget for providing a "Continuous Mode" enable/disable setting. I used the "button" widget for the "SCAN" command, but the best widget on the GUI is the double-ended slider which serves a dual purpose. It always represents a 1MHz to 30MHz range but a highlighted band within the slider shows the region that will be swept. This region is either reported back from selecting one of the canned bands in the Band dropbox, or the slider can be used as an INPUT where you can select the swept band limits. In addition, you can use the slider to pan and zoom within the graph during or after a sweep.

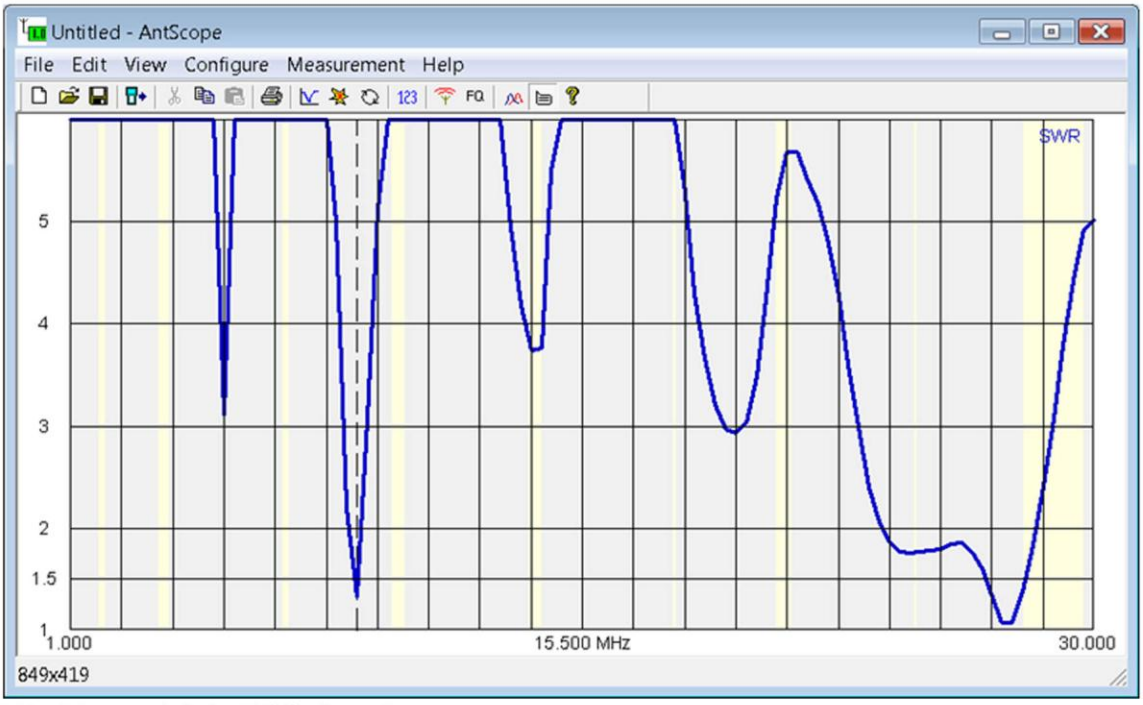
Performance

The results compare favorably with a RigExpert AA-520 commercial analyzer. The absolute values of the reported VSWR from the K6BEZ Analyzer appear to be a little off, but most of the utility from this inexpensive instrument is obtained simply as a dip-meter at the frequency of interest, and the "ballpark" SWR provided by the sweeper. The precise SWR is of secondary concern (to me), since decent ballpark accuracy has been established via the comparison plots on the next few pages.

1-30 MHz sweep of KN5UTE 54ft Balanced Doublet Antenna

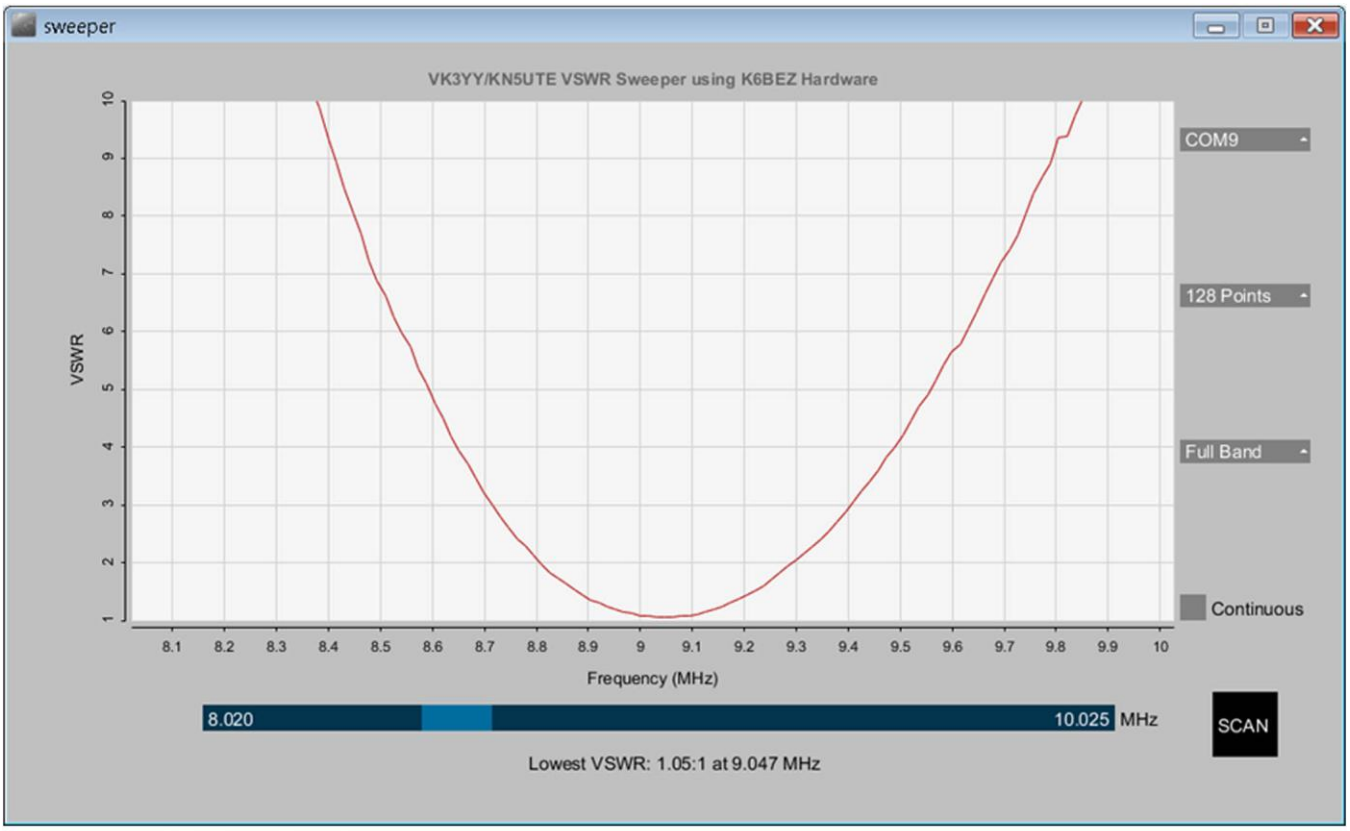


K6BEZ Analyzer

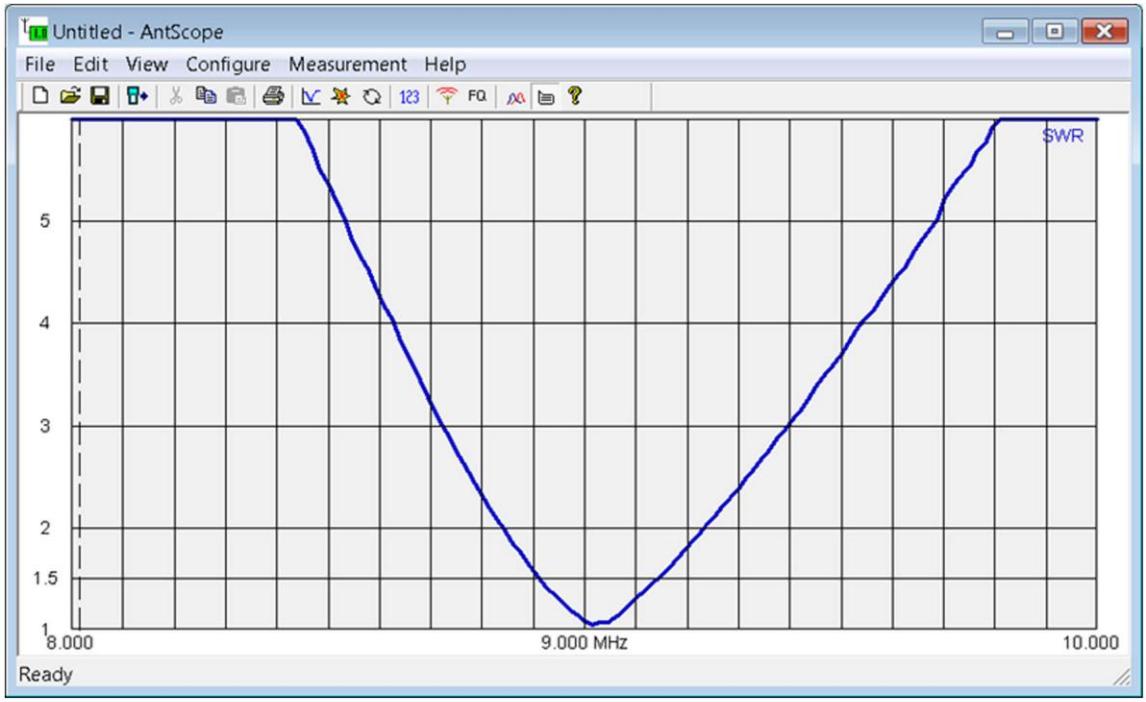


RigExpert AA-520 Analyzer

8-10 MHz sweep of KN5UTE 54ft Balanced Doublet Antenna

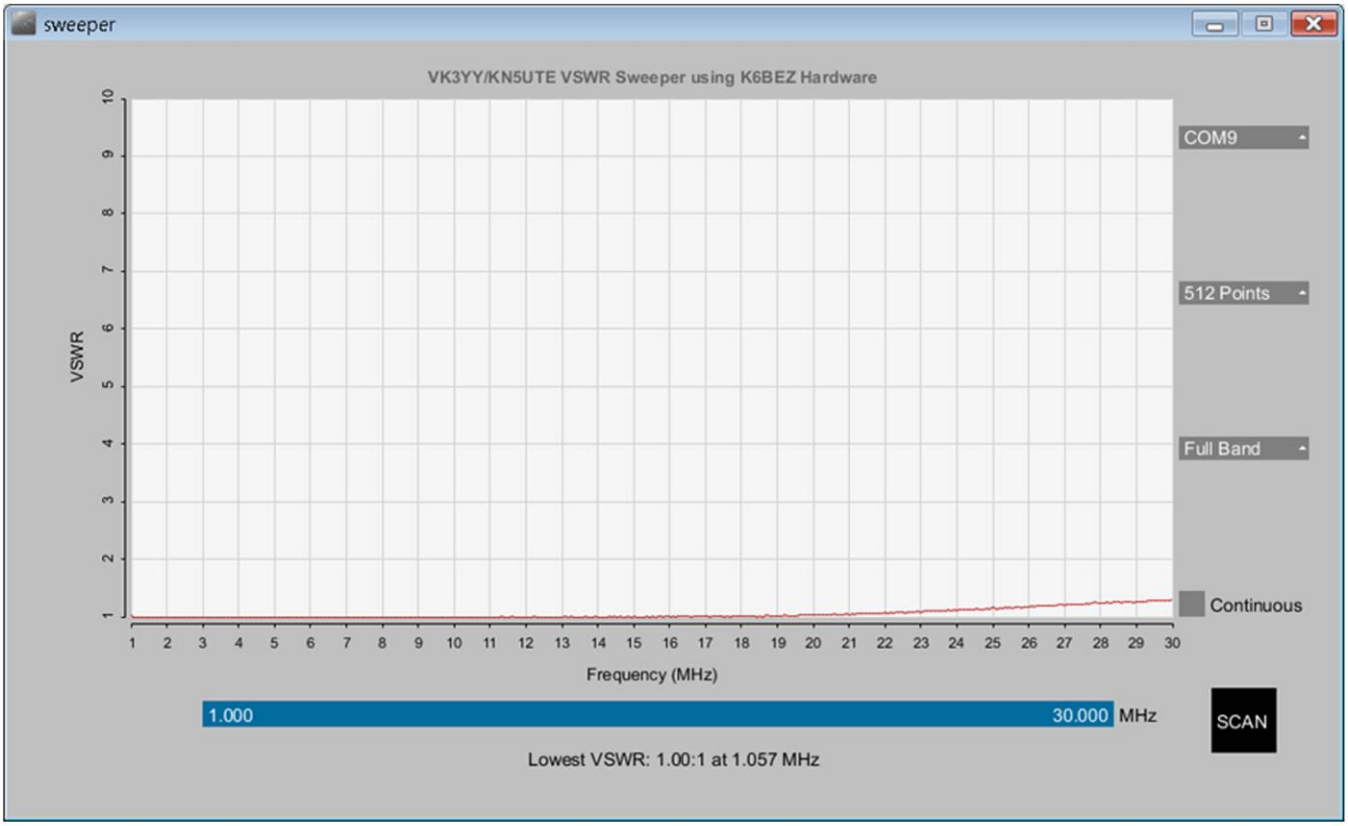


K6BEZ Analyzer

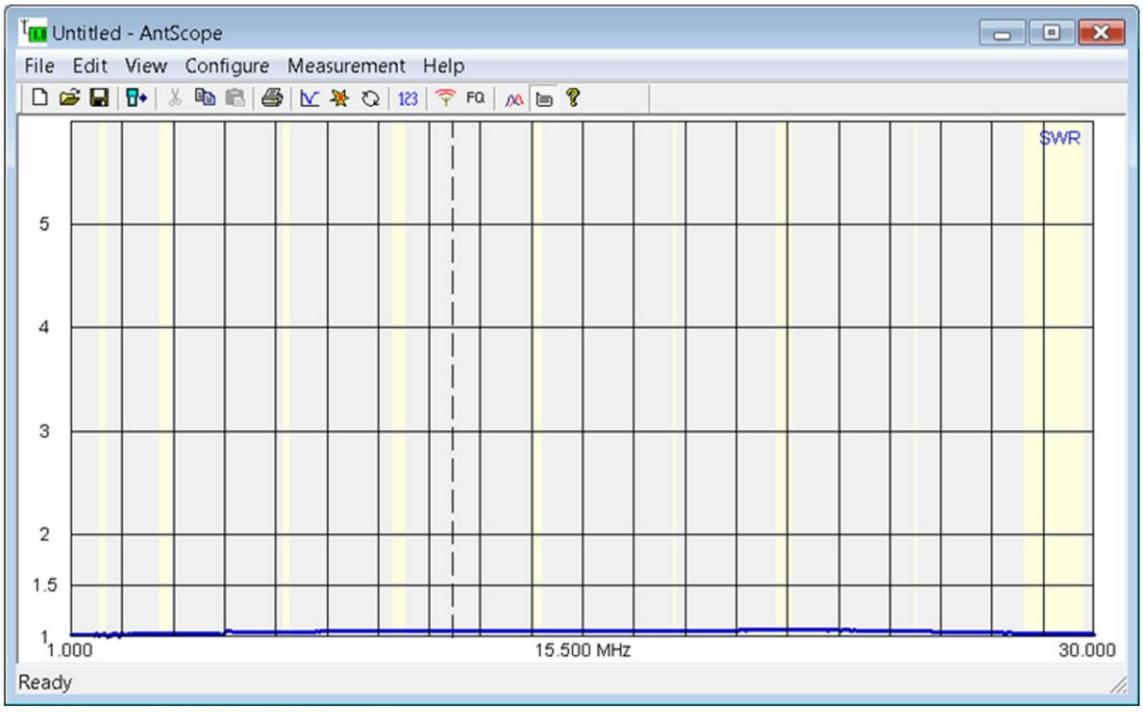


RigExpert AA-520 Analyzer

1-30 MHz sweep of MFJ Dummy Load



K6BEZ Analyzer



RigExpert AA-520 Analyzer

Usage

The way I intend to use the sweeper is to set the band of interest, and enable continuous scanning. I can then adjust my tuner controls until I achieve lowest SWR at the intended transmit frequency. If I were to add to the GUI, I would include the ability to command the DDS to a single un-swept frequency, and provide a bar graph showing SWR at that frequency which would reflect the real-time settings of the tuner controls. The Arduino controller code already supports a single frequency output, so this change would simply be change to the PC program.

Conclusion

It was inexpensive, fun to build, I got to code a little, I learned some new stuff along the way, and it provides sufficient accuracy for my purposes. I would recommend it to others who are looking for a similar capability, and are comfortable with a modest construction challenge. There are several similar designs on the web at present, some kits are available that reduce the already modest risk factor.