The W8TEE and K2ZIA Antenna Analyzer

by
Jack Purdum, W8TEE
Farrukh Zia, K2ZIA

This manual was originally written for members of my club, the Milford Amateur Radio Club, many of whom had never built a kit before. I also kitted the parts for the antenna analyzer (AA), something I will NEVER do again! My appreciation goes out to everyone who creates a bag of parts for a kit. Because of that background, some of the instructions apply to parts of that kit, but you are required to buy your own parts. Once acquired, you can use this manual to build the AA.

The W8TEE/K2ZIA antenna analyzer (AA) has the following features:

- VSWR measurements for continuous scans for any amateur band frequency between 1-30MHz.

- Predetermined US band edges for quick entry of scan start and end points – all lower and upper band limits serve as default scan points, fully adjustable with a simple turn of the encoder. Band edges can be changed for other countries.

- Large (3.5" x 2.125") color TFT display for scan plots – a 262,000 color display with 480x320 pixel resolution. Compare to other higher priced units with 128x64 resolution.

- Save scan data. An optional 2Gb SD card allows you to save over 9000 scans! Others only allow limited scans (e.g., 10) that are save in memory. Turn the machine off and they are lost. Not with the LAA!

- Scan data export. The scan data are saved in the popular CSV format and can be exported via the USB port for use in other programs (e.g., Excel, graphics package, text editor, etc.)

- Scan overlays. Run a scan and save it to the SD card. Now make a change to the antenna, and run another scan and immediately overlay the previous scan plot to the current scan plot to assess the impact of your change on the antenna.

- 100 scan point resolution regardless of scan spread – compare to other analyzers that use only 12 scan points.

- Fast scans, typically less than 5 seconds for a 100 step scan – compare to 30 seconds with fewer scan points other units.

- Portable use with 9V battery or use a 9V wall wart when grid power is available; perfect for in the field or home use.

- Simple two control user interface...and one of those is the power switch! This means easier construction.

- High quality PCB that simplifies connections to the Arduino Mega2560 Pro Mini board and the TFT display.

The purpose of this manual is to help you build your AA. We also present some pre-construction ideas that may help you get organized for the assembly and make the construction easier.

Step 1. What you need to complete the kit

To complete the AA, you will need to purchase the parts that are given in the list of materials found at the end of this manual. In addition, you will also need:

1. A soldering station or iron. Usually a 25-30 watt iron is sufficient, but a soldering station is nice to have. It has a holder for the hot iron, a sponge for cleaning the tip while you solder, and an adjustable temperature control. (Most of the time I run at the hottest temp.) Figure 1 shows the station I use which is available on eBay (#271878518662) for about $25.

Figure 1. Solder station

2. Tools. Something to strip wires, needle nosed pliers, small screw drivers (flat head and Phillips). If you use a box cutter to trim insulation from wires, try not to nick the wire in the process. For clipping component leads, I use toenail clippers. ***Warning***: clipped leads know how to travel at the speed of light when clipped. Wearing protective glasses is a good idea.

3. Solder. Use rosin core solder only and the thinner, the better. I use .022" silver solder (62/36/2) which is a blend of tin, lead, and silver. I bought mine at Radio Shack, but it's getting hard to find. (See bottom-left of Figure 3.) Check eBay. Unless you want to buy a pound of the stuff, it's useful to sort the eBay list by price.

4. Magnifying glass. (Also shown in Figure 3.) Even with good eyesight, it makes reading resistor band colors and capacitor numbers easier. Good lighting is important when reading color bands. Purple and blue look a lot alike in poor lighting.

5. A digital multimeter (DMM). Don't chintz here...get a good one. I really like mine, an AideTek VC97. (See Figure 2, eBay #290513474085). You can buy this one on eBay for less than $30. It can be used to check resistance, voltage, amperage, capacitance, and transistors and comes with a nice case for storage. It even has a temperature sensor. Also, if you leave it unattended for a few minutes, it beeps to remind you to turn it off.

Figure 2. The Aidetek VC97 multimeter.

What follows is a list of things that are nice to have, but not required

1) Solder sucker (#351065197104) or solder wick (#290768709802). Used to remove solder from a connection if a solder error is made (e.g., soldering the wrong resistor in place). If you never make a mistake, you don't need it.

2) PCB holder, about $10.00 (#401112964765). Makes it easier to hold a PCB in place while mounting and soldering components. I bought one on eBay because I build a lot of kits. See Figure 3. You can rotate the board easily after mounting the parts for soldering the components on the "under" side of the board. You can also see a dirty old towel under the holder as sometimes I drop a part and it's easier to find on the towel, plus the parts don't bounce when they hit the table. You'd be surprised how hard it is to see an 1/8W resistor on certain types of carpet. (I'm old and it happens.)
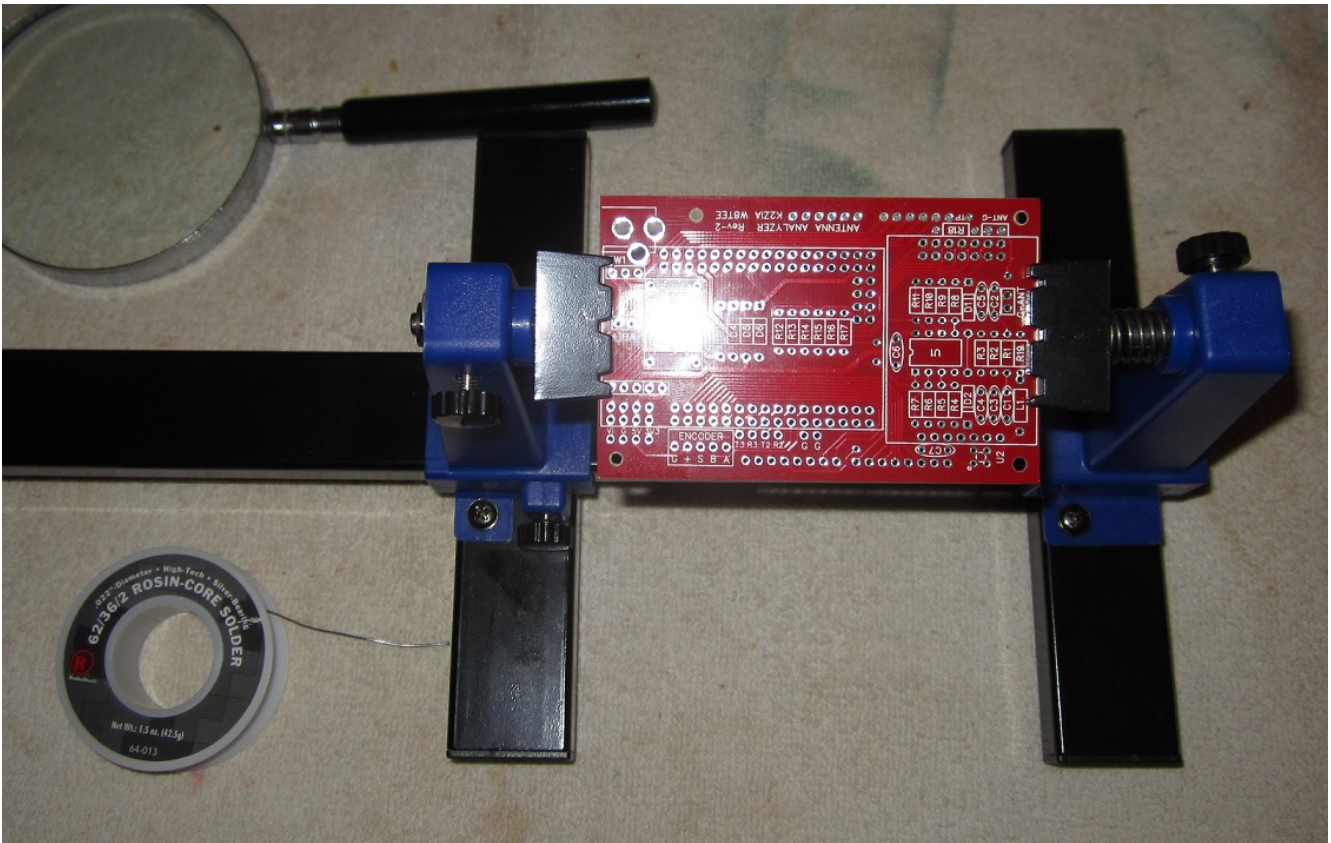
Figure 3. PCB holder.

3) I recently purchased LCR-T5 multifunction tester. It cost me $20 and is one of the best eBay purchases I've made. It can test resistors, capacitors, inductors, diodes,

Figure 4. LCR-T5 multifunction tester

transistors...almost any component you're going to use when building a kit. While I can read those numbers on a capacitor with a magnifying glass, testing the part with one of these shows what its value actually is (some kits have caps that are off by 50% or more) regardless of what the number says it *should* be.

4) My next best purchase recently was a magnifying glass that has a push button switch on

Figure 5. Magnifying glass with 6 LEDs.

the handle and has six bright LEDs built into it. I don't know about battery life yet, as I've only had it for a few months. I also use it to find parts that I've dropped onto a multi-color carpet that in my shack. It's amazing how a resistor simply disappears on this carpet. This glass helps me find it. Worth much more than the $7 I paid for it.

Step 2. Getting Ready

When you do a parts inventory, it's worthwhile storing the parts so they are easy to see. I often use a sheet of packing styrofoam to stick the components in, even though it's not a good idea for static-sensitive parts. My shack is in the basement and I could rub two cats together and not get a spark, so
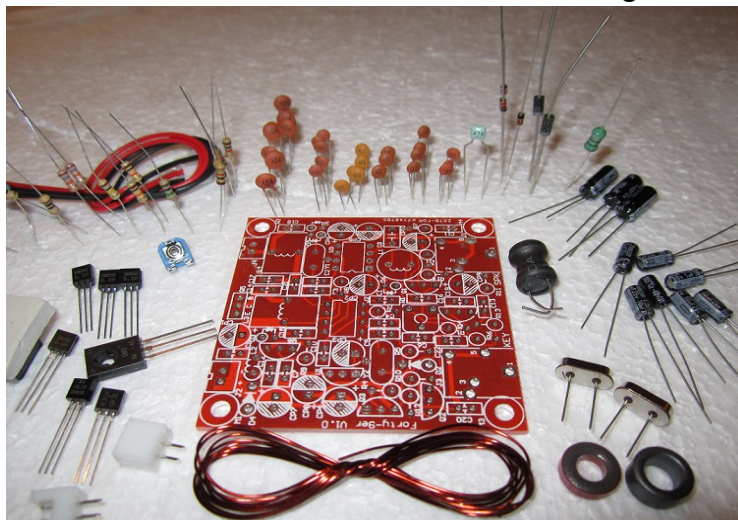

Figure 6. The Forty-9er kit before assembly.

I don't worry about static issues too much. (Figure 6 shows the parts for the Forty-9er transceiver kit prior to assembly.) Note how some capacitors are "in a line". That means they are all the same value. I also do this for resistors.

Another good "parts holder" is an old egg carton. I've also seen builders take a 4" strip of cardboard, bend it into a 'U' shape, and place components in the corrugation channels. Whatever method you use, the point it to make it easy to identify and reach the parts.

As you place a component onto the PCB (or otherwise add to the project), make sure you check it off your parts list once it's in place. Obviously, you want your "parts holder" to be empty when all of the items on your parts list are checked off.

Figure 7 shows several of the parts in the project so you can tell them apart. Note how the
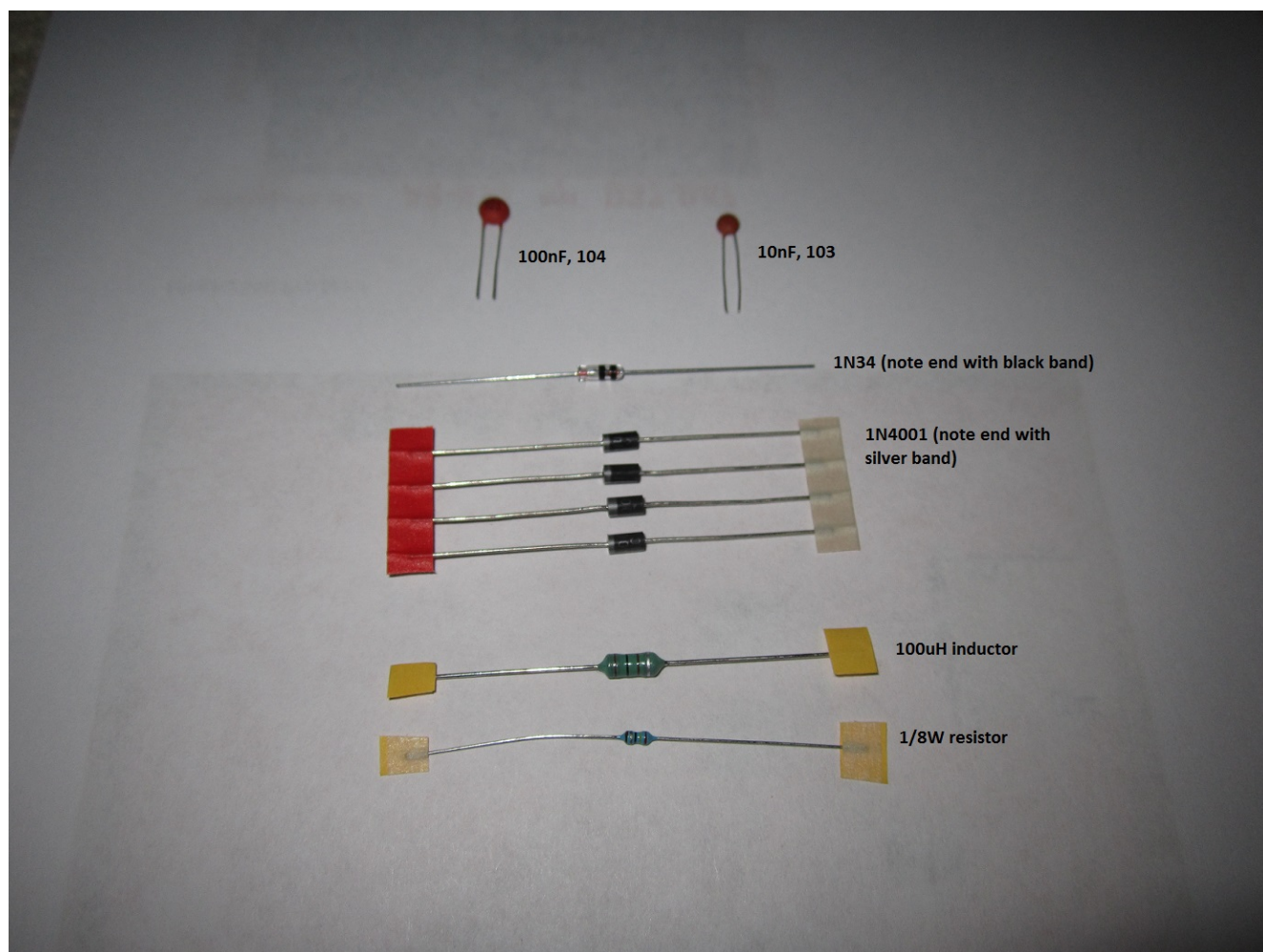

Figure 7. Some parts from the AA

1N34 and 1N4001 diodes have bands painted on them. All of these diodes must be placed on the PCB so the band matches the band on the diode. (The band on the PCB looks like a single line, but it actually forms a band with the end outline of the diode on the PCB.)

Figure 8 is a picture of the Printed Circuit Board, PCB, that is used with the AA. The figure shows the connections to the BNC antenna connector made from the right side of the PCB. However, if you look closely at Figure 8, you can see a second sets of antenna takeoff points in the upper-right corner of the figure, just to the left of the mounting hole. Either set can be used. Your choice may be influenced by the way you place the PCB in your enclosure.
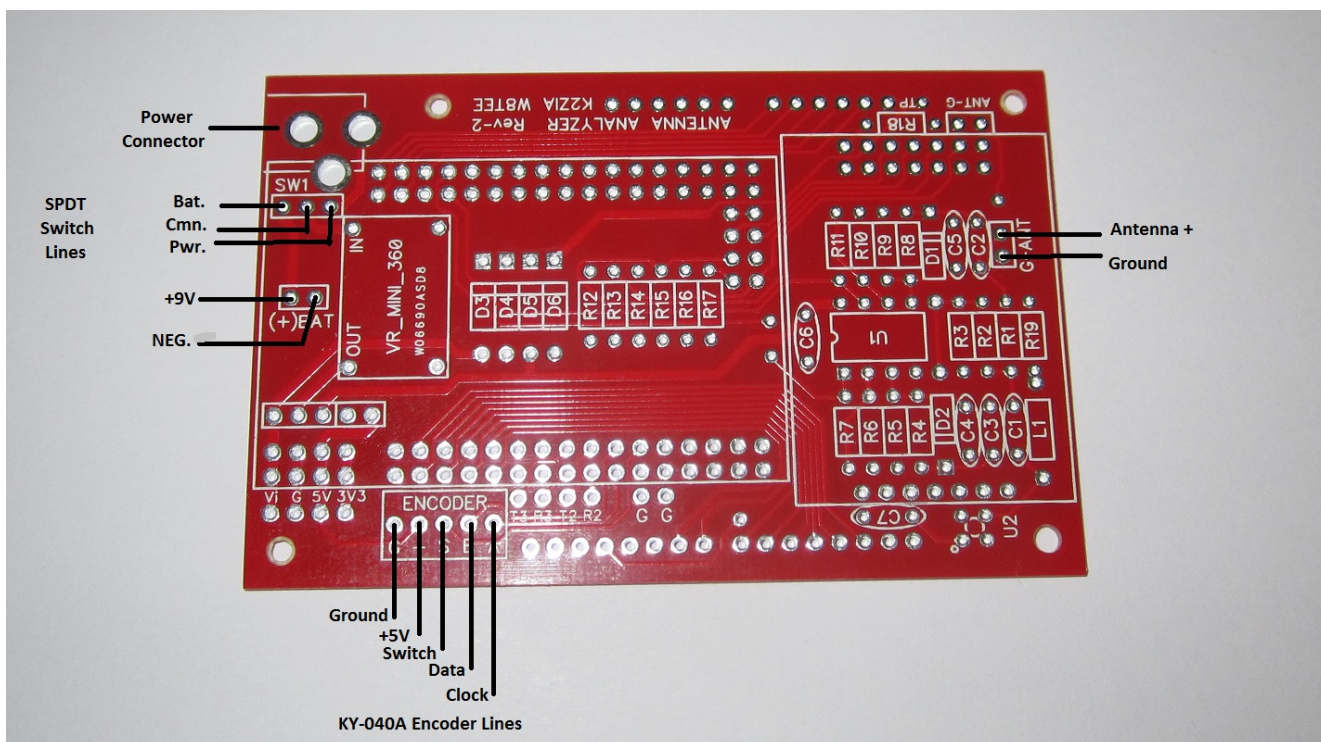
Figure 8. PCB with connections to external components shown.

Figure 9 is a closeup of the TFT display on the bottom and the PCB and its supporting boards in place.
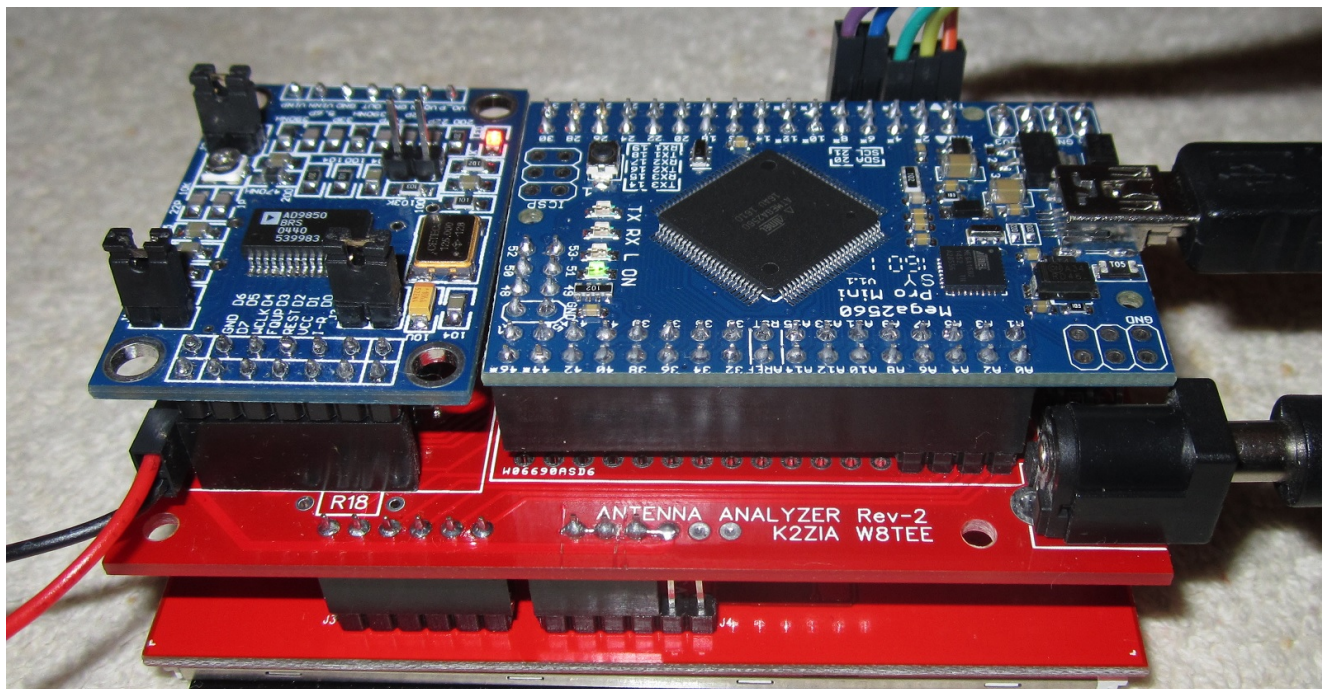

Figure 9. Closeup of TFT display, PCB, DDS, and Mega2560 Pro Mini boards.

In Figure 9, you can see the antenna connection running off to the left, the encoder connections near the top right of the figure, the wall wart power connector on the bottom-right, and the USB connection to the Mega2560 Pro Mini near the mid-right corner of the figure. (Later on, we decided not to use the

power connector on the PCB. Details are provided later in this manual.) The TFT display "plugs into" the AA PCB via pins on the display and sockets on the PCB. Note in Figure 9 (near the bottom-center) that two pins on the TFT display are not socketed to the PCB, as I didn't have any more 6 pin headers. . This is ok as those pins are not used in this project.

Before we start adding components, you should read through *all* of this manual *at least twice*. That will give you a good idea of how things will go together. Also, it is easier to drill the holes in the case that will later secure the PCB and display if it is done before anything is on the board. That way, you can lay the PCB flat on the bottom of the case for marking the mounting holes. See Figure 30.

Okay, it's time to start building.

Step 3. Adding the resistors

Buying Parts

Depending upon where you buy your parts, it is often cheaper to buy a resistor assortment than buy them one at a time. One local parts store sells resistors at $1.50 for two resistors. I buy a resistor assortment from a GA eBay store (#192184944181) 130 values @ 20 each for $10. I buy my capacitors the same way (#321891053652), 1000 for @ 50 different values for $7. These stores bag/label the parts which some vendors do not do. It's worth having the parts labeled. Some of the resistors are "taped" together, as they were cut from a reel of like values. All of the resistors are 1/8W, so you don't need to worry about that characteristic. However, you do need to make sure that you select each resistor with the correct resistance before placing it on the PCB.

Tayda Electronics has good prices and a distribution warehouse in the US. However, while shipping charges are fair, it's best to buy as many parts as possible at one time. Splitting an order with a friend is always a good idea, too.

The resistor and capacitor values used in this project are not that critical. Being off 10% for the resistors or capacitors shouldn't make any difference. If you build more critical circuits (e.g., a tune circuit), you may need more precise values.

Several recommendations:

1. Follow the sequence for mounting components as suggested below. Sometimes place one part is more difficult if you have already mounted some other part first. The sequence is done to minimize such issues.

2. Don't mount all resistors at one time. Do them in small groups of no more than 6 at a time.

3. I prefer to mount all of the resistors with the same value at one time. In this case, I will mount the resistors in the same sequence shown in Table 1.

4. If you have a resistor assortment, each group is part of a "tape" which holds a label for that resistor's value. Clip the resistor leads near the tape holding them together, thus preserving the label for later use.

5. Measure each resistor before placing it on the board. Same for capacitors.

6. Reread number 5.

7. Consistently mount all resistors so the color bands can be read left-to-right. Bend the leads in a

'U' shape before placing on the board, fan the leads on the back side of the board.

Table 1. PCB resistors

Ohms	Color Code	Schematic Part Number
10	Brown Black Black	R19
51	Green Brown Black	R1, R2, R3, RL*
1K	Brown Black Red	R6, R10, R13, R15, R17
2K	Red Black Red	R12, R14, R16
10K	Brown Black Orange	R4, R7, R8, R11
100K	Brown Black Yellow	R5, R9

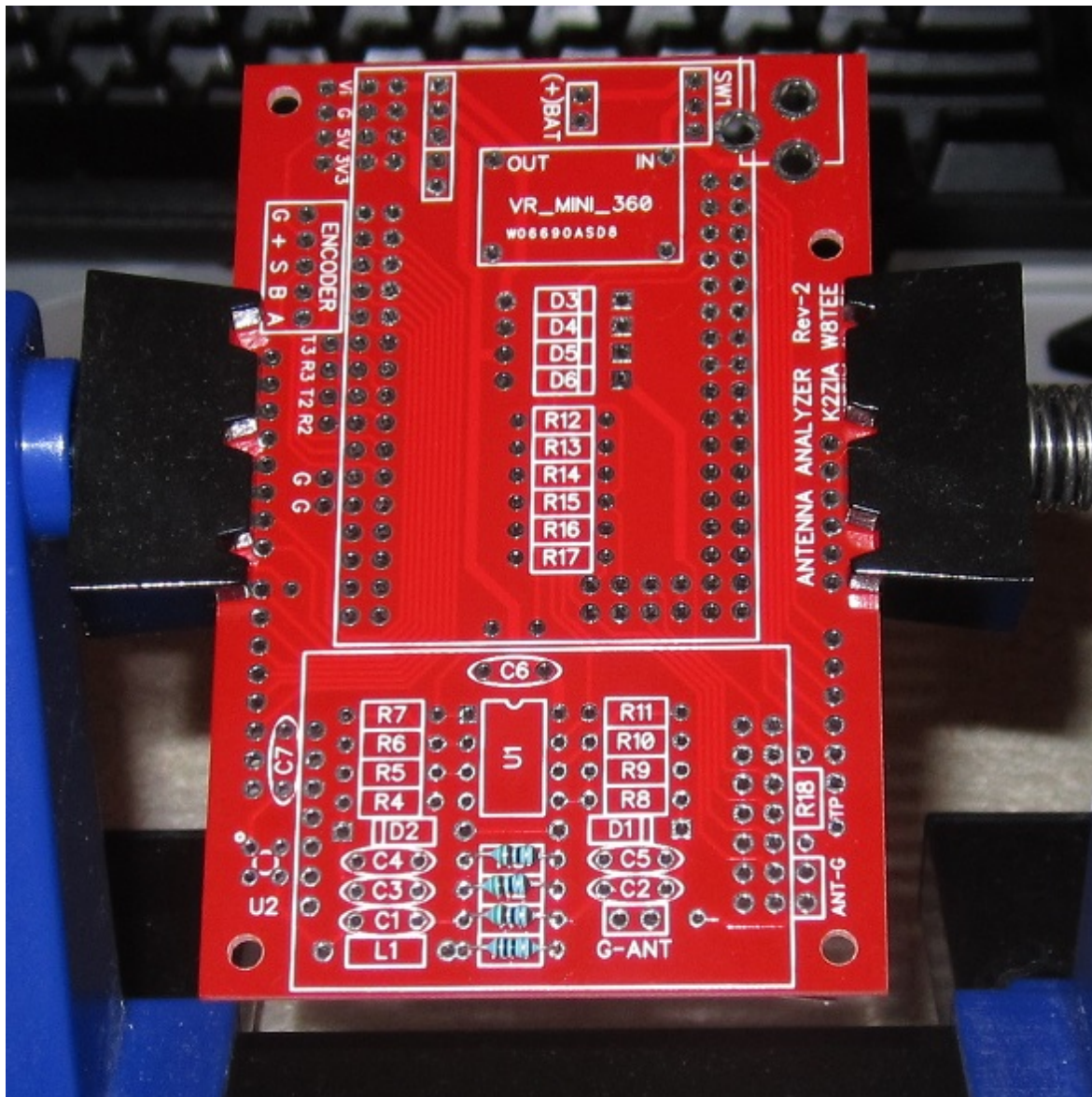* This resistor is not mounted on the board, but may be used as an antenna "dummy load" while testing.

Figure 10. Resistors R1, R2, R3, and R19 in place.

In the schematic (Figure 33, at the end of this manual), the values for the resistor values are ***not*** correct, but their ID numbers are. For example, resistor R12 is shown as a 3.3K resistor in Figure 33, but it is actually a 1K resistor as shown in Table 1. Also, we had to substitute the MSA-0386 op amp at U2 with an MAR-3SM+ component. (The MSA-0386 is getting hard to find.) The schematic will be changed, but for now some of the values are incorrect but they are correct in the Table.

Figure 10 shows several the resistors in place and Figure 11 shows them after they are soldered in place. Before I clip the leads, I "strum" each resistor lead with my thumbnail. If it gives an almost a musical note, it's a good connection. If it gives a "thunk" sound when strummed, it's probably a cold solder joint and not a good connection. Re-heat the connection and test again. Cold solder joints are probably the most common reason for the failure of a project.
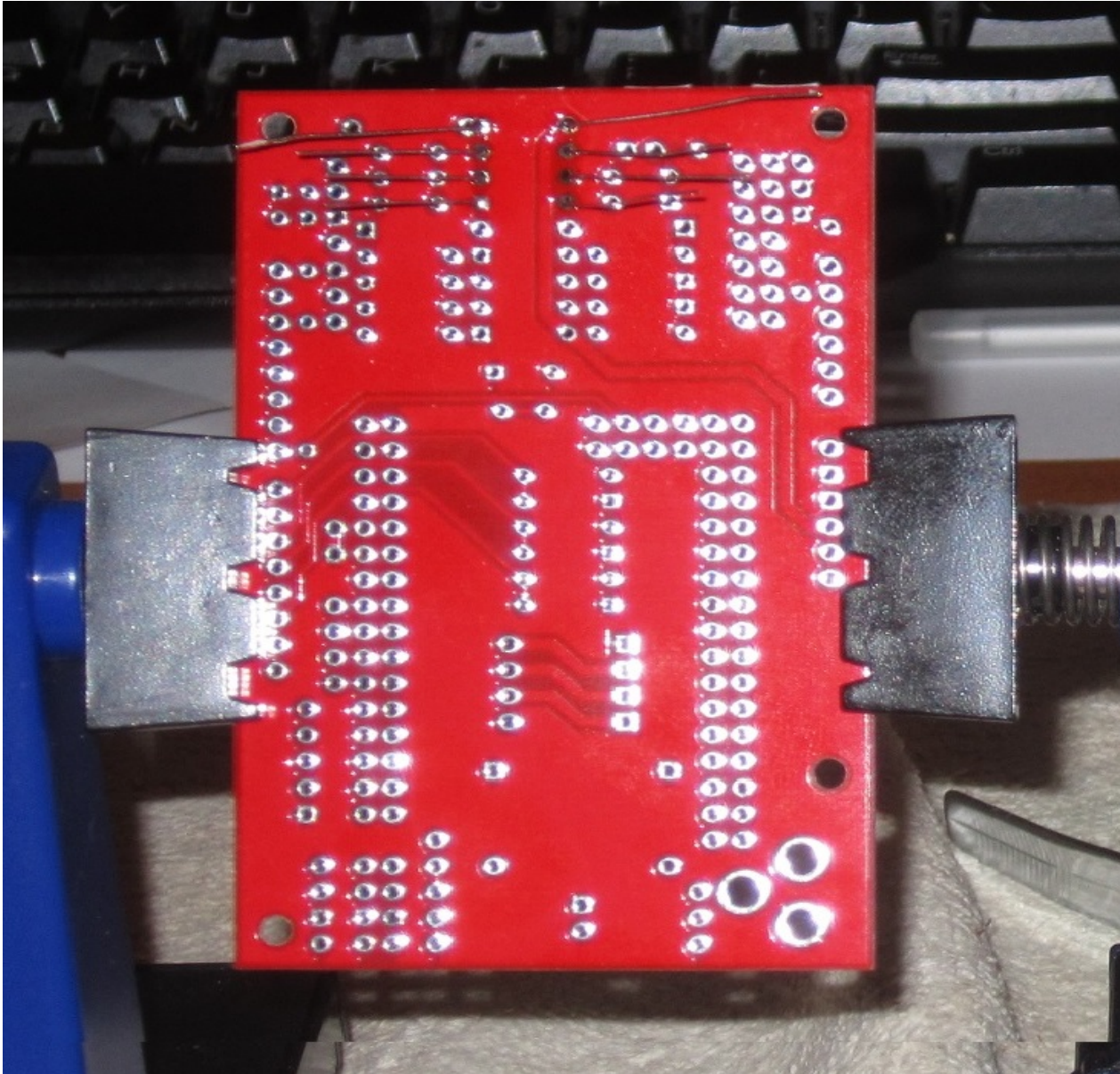

Figure 11. The flip side of the board after soldering

Use whatever tool you have chosen to clip the resistor leads as close to the board as possible. Figure 12 shows what the clipped leads from Figure 11 should look like. You can barely see where they are soldered to the board. Make sure you clip them as short as possible, as leads that are even a little bit

longer could bend directly above another solder-through hole. This could produce a short if you don't see it. Also, as the solder melts, it leaves a slightly sticky residue on the board. That's not a problem, but sometimes small pieces of clipped leads stick to the residue and could cause a short. Make sure you remove all of the excess lead clippings.

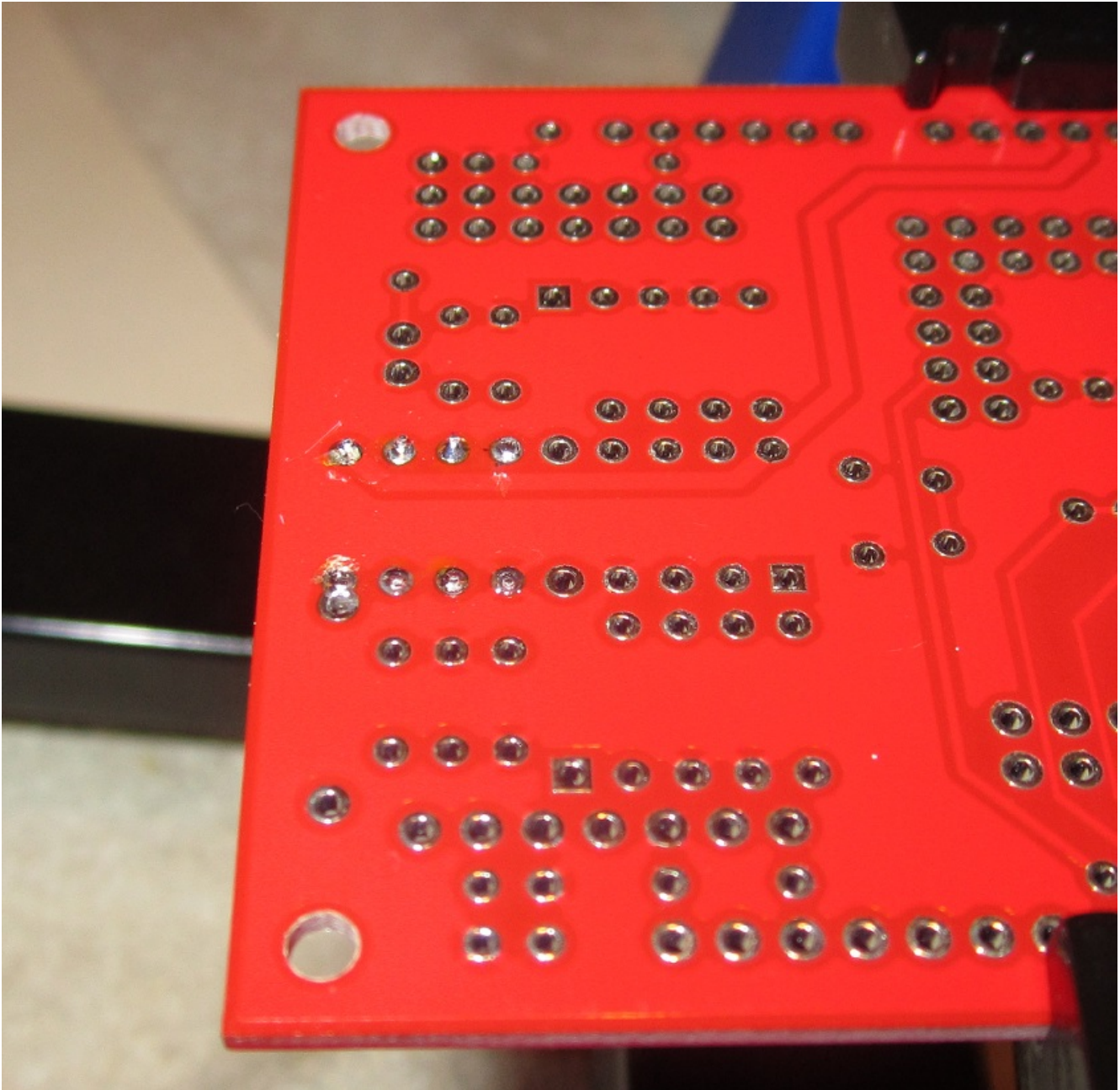*Figure 12. Resistor leads after testing, soldering, and clipping.*

Proceed with soldering the remaining resistors in place and clipping their leads.

Capacitors

There aren't many capacitors to mount, so just do them as two separate batches, measuring each before placing it on the board. Each capacitor has a standard capacitor number stamped on it. I try to place the

capacitor on the board so I can read these standard capacitor ID numbers easily. (Elecraft has a great little table of the standard cap numbers: http://www.elecraft.com/Apps/caps.htm.) Sometimes neighboring components make it difficult to read their values, so a little thought about the numbers when putting them on the board can help. The 10 nano farad caps are the smaller of the two. Soldering and clipping their leads is done the same way you did the resistors.

Table 2. Capacitors and their standard numeric values

nF	Standard Number	Schematic Part Number
10	103	C4, C5
100	104	C1, C2, C3, C6, C7

Diodes

There are two types of diodes in the kit as shown in Figure 7. The look very different so it should be easy to tell them apart, plus it's easy to read the numbers for the 1N4001 diodes.

Diodes, transistors, and most semiconductor devices get a little cranky when you apply too much heat to them. Because of this, I have a sequence I use when soldering them to a board. First, I try to mount them so I can read their numbers. (For this board with such a low diode mix, it probably doesn't matter much.) I bend their leads in the same manner I do for resistors. However, when it comes time to solder them, I only solder one lead of a diode at a time. I move to the next diode and solder it, leaving one lead unsoldered on the first diode. I do this for all of the diodes on the board. When I'm done with this process, all of the diodes will have one lead soldered in place.

Now I go back to the first diode and solder the remaining lead in place, then move to the next one. Using this approach gives the diode some time to cool off before the second lead is soldered. When I'm done, I again strum each lead just to make sure there's no cold solder joint (or I didn't forget to solder a lead). I've told other builders about this and they assure me it isn't necessary as long as you don't dawdle while the soldering tip is on the component. After hearing this, I ask them: "Have you ever had a diode fail from over-heating?" Most have. I've never had one fail. *Quod erat demonstrandum.* (Latin for: "Put that in your pipe and smoke it!")

Table 3. Diodes used on the PCB.

Diode type	Schematic Part Number
1N34	D1, D2
1N4001	D3, D4, D5, D6

If you look closely at Figure 10, you will see a small solid line to the left of "D2" and the same type of line to the right of "D1" on the PCB. These markings show the orientation of the diode. The "band end" of the diode marks the cathode and the unbanded end is the anode. This is shown in Figure 13.

You want to make sure the band painted on the diode aligns with the band silk screened on the PCB. The 1N34 has two bands painted on it, with one of those bands at the very edge of the diode. That is the "banded" cathode end of the diode. Both D1 and D2 have the cathodes facing towards the edges of the board. The 1N4001 cathodes are clearly marked with a silver band. You can check this with your DMM by placing the scale on Resistance and the positive probe on the cathode and the negative probe on the anode. You will see a very high resistance. Now reverse the leads and the resistance will be much
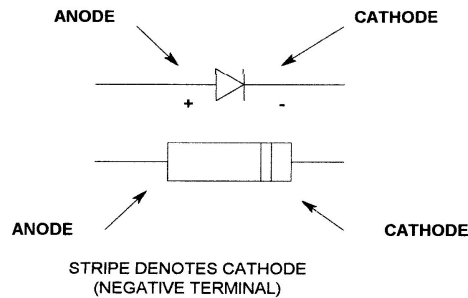
lower.

ANODE   CATHODE

ANODE   CATHODE

STRIPE DENOTES CATHODE
(NEGATIVE TERMINAL)

Figure 13. Diode markings.

The 1N4001 diodes all face the same way, with their cathodes closest to the right edge of the board in Figure 10. Solder them in place using the same techniques.

Miscellaneous Parts

There are a number of other parts mounted to the PCB. We consider those in this section.

8-Pin IC Socket

The IC socket has a small notch in the center of one end of the socket. If you look on the PCB for U1, you will also see a small notch silk screened on the board. Place the socket on the board so the two notches line up. If you look on the LM358 chip, you will see a matching notch in the chip. (Some IC's have a small dimple next to pin 1.) These notches must line up when you seat the chip in its socket.

Place the IC socket on the PCB, making sure the notches line up. Flip the board over and bend a socket pin at opposite ends of the board to hold the socket in place while you solder all 8 pins. Do *not* place the chip in the socket at this time.

Inductor

As shown in Figure 7, the axial inductor looks a little bit like a resistor because it, too, has bands on it. However, compared to an 1/8W resistor, the inductor looks like it's on steroids so it's pretty easy to tell them apart.

The 100uH inductor (L1) is mounted towards the bottom of the board, to the left and below the IC socket. I mount it with the black band on the left side, but the leads are interchangeable, so it really doesn't matter. Solder it in place.

Power Socket

Do **not** mount the power socket on the board. Directions are given later for mounting it on the case. I made this change because of the difficulty of getting the on-board connector to properly align with the wall wart connector. The TFT display made this alignment difficult. For now, do not mount the power connector on the PCB. I'm not going to re-shoot all the photos, so just pretend the connector is not there.

You can always come back to this point in the instructions and mount the power connector on the PCB if you decide that's best. Directions for mounting the power connector off board are given towards the end of this document. For now, I suggest leaving the power connector off.

Figure 14. Resistors, capacitors, diodes, IC socket, and power connector in place on PCB. Note we later removed the power connector, hence it is X'ed out in this figure.

Header Pins

Header pins are used in conjunction with Dupont prototyping wires so that removing the boards is more easily done. These items are shown in Figure 15. Each of the wires has a plastic female connector on it and the pins form the male connection points. The pins are used to attach the encoder, antenna, power switch, and battery to the PCB. I prefer to use pins on the board for the power switch, but solder the Dupont wires to the three positions on the switch. To do this, I clipped the connector off one end of all three wires that connect to the switch. The connector end slips onto the pins located at SW1. (Just to the left of the power connector in Figure 14.) Using the Dupont wires makes it easier to disconnect things if you need to repair something.

If you look just below C2 in Figure 14, you will see connection points for the antenna (G = ground, ANT for center pin of the BNC connector.) However, about an inch to the right of that point and
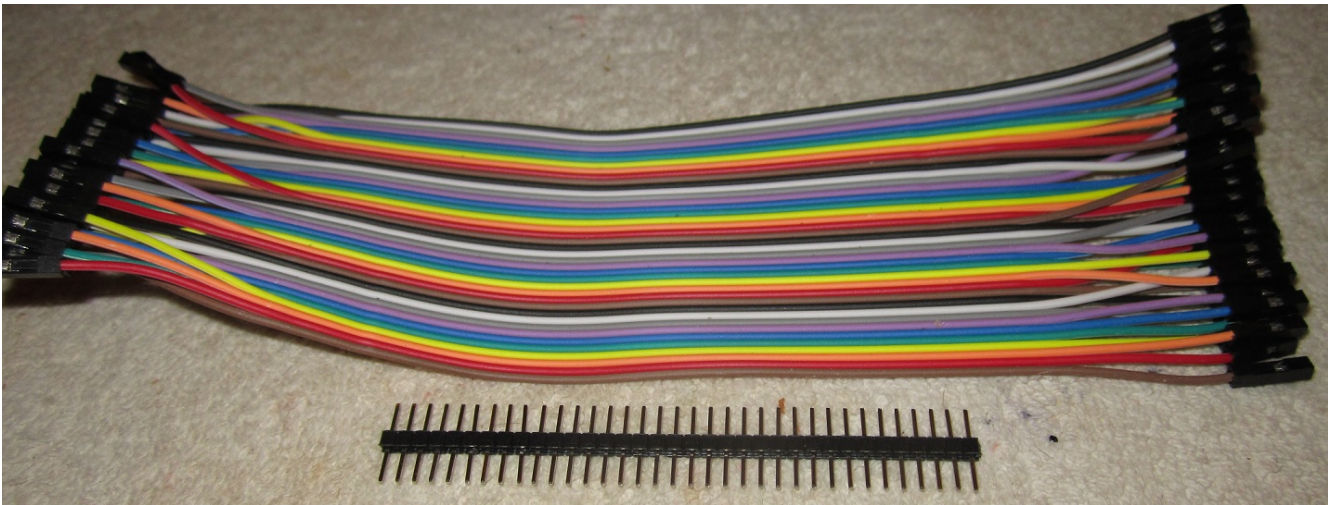
Figure 15. Dupont prototyping wires and header pins.

towards the edge of the PCB in Figure 14 you can see a second set of takeoff points for the antenna connection. This gives you two options for the antenna connection which may be influenced by your case selection.

The Encoder Pins

During the first iteration of this manual, I thought it would be a good idea to push the plastic collar that holds the pins downward to make the connecting pins longer. Not a good idea. As it turns out, doing that makes the pins less mechanically stable. Figure 14 shows how the Mega 2560 Pro Mini socket pins are placed on the board. (Note we are using a PCB from another project, but it also uses the AD9850
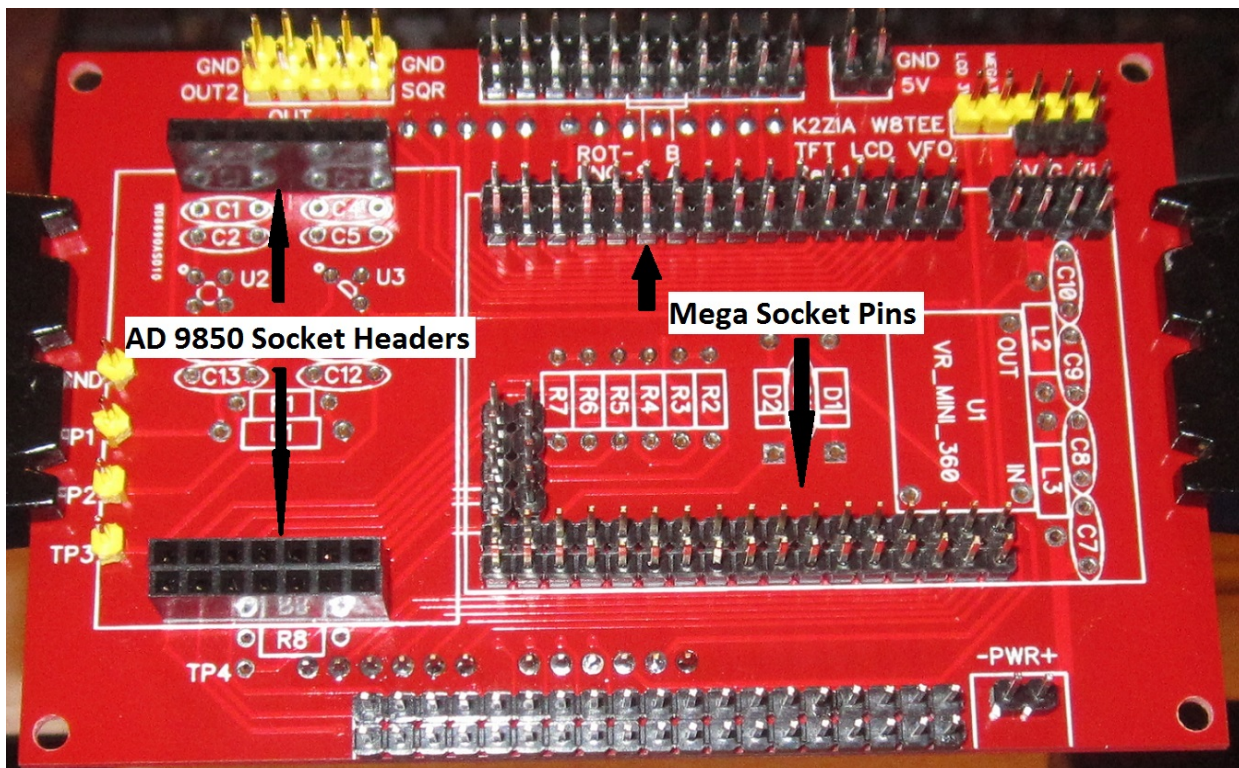

Figure 16. Mounting pins and sockets on PCB.

and Pro Mini.) You want to place the shorter lengths in the PCB holes, leaving the longer lengths available for the Dupont connecting wires or sockets. Do not break all of the pins apart and try to solder

them individually. Instead, break off a group of pins that will completely fill a row on the board. Doing this adds rigidity to the Mega.

Figure 17 shows how I use a piece of Scotch tape to hold the header pins in place while soldering them. The tape holds them in place so I can flip the board over and solder the short pins to the PCB. The tape doesn't have to be "tight"; just enough to keep them from falling out when you flip the board over. Try to make sure the plastic collar sits flush with the PCB surface.
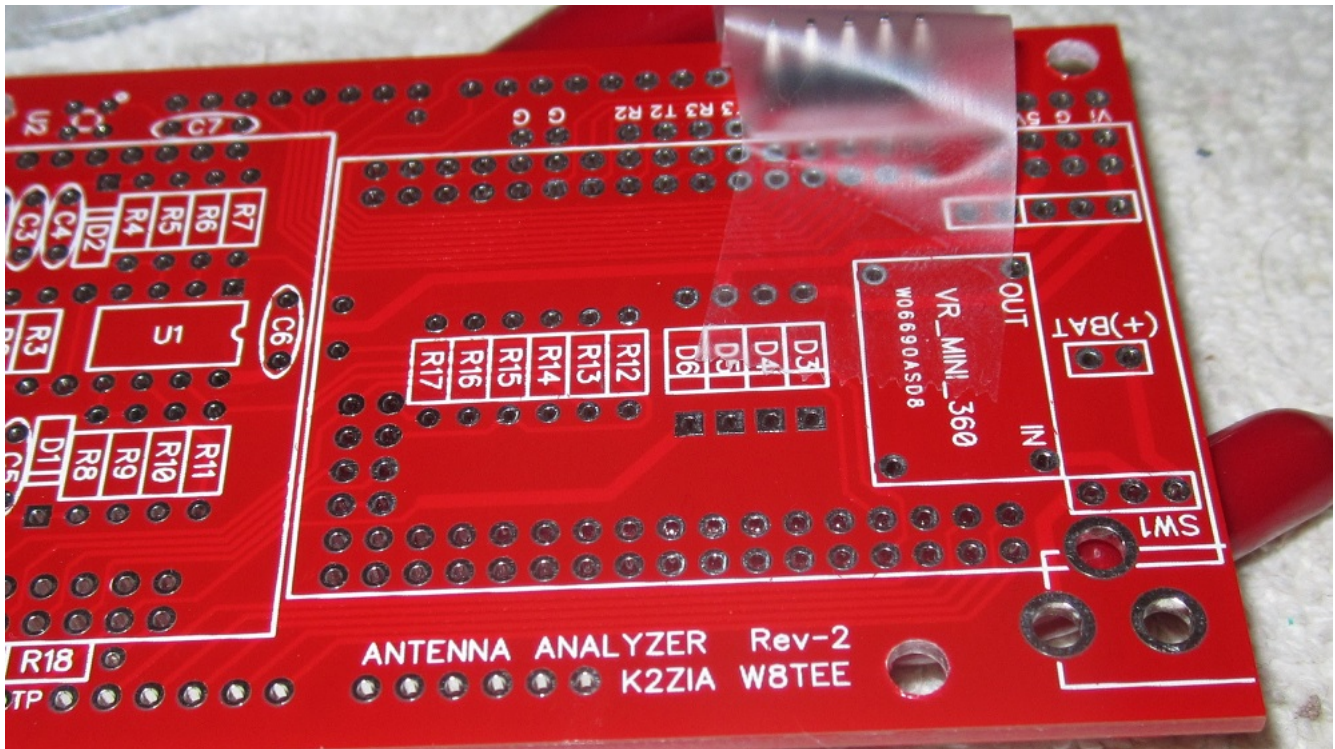

Figure 17. Holding the pins in place for soldering.

Repeat this process for the antenna, battery, and switch connection points.

Because the Mega2560 and AD9850 boards sit on top of the PCB, it is necessary to bend the battery, antenna, and SW1 pins downwards toward the PCB so they can clear the two boards on top. (The encoder pins are "outside" of the two boards.) For now, bend the pins down to about a 45 degree angle. You can adjust this later when the two top boards are in place. See Figure 18.

Once you have all of the pins soldered on the board, you could mount the Mega 2560 Pro Mini and try to compile the Blink sketch that comes with the Arduino IDE. Connecting the USB connector from your PC should cause the Mega power LED to light up and accept programs. If the onboard LED does not light up or the USB serial I/O LEDs to not light up when transferring a program to the MEGA, check to make sure all of the pins have been properly soldered to the PCB.
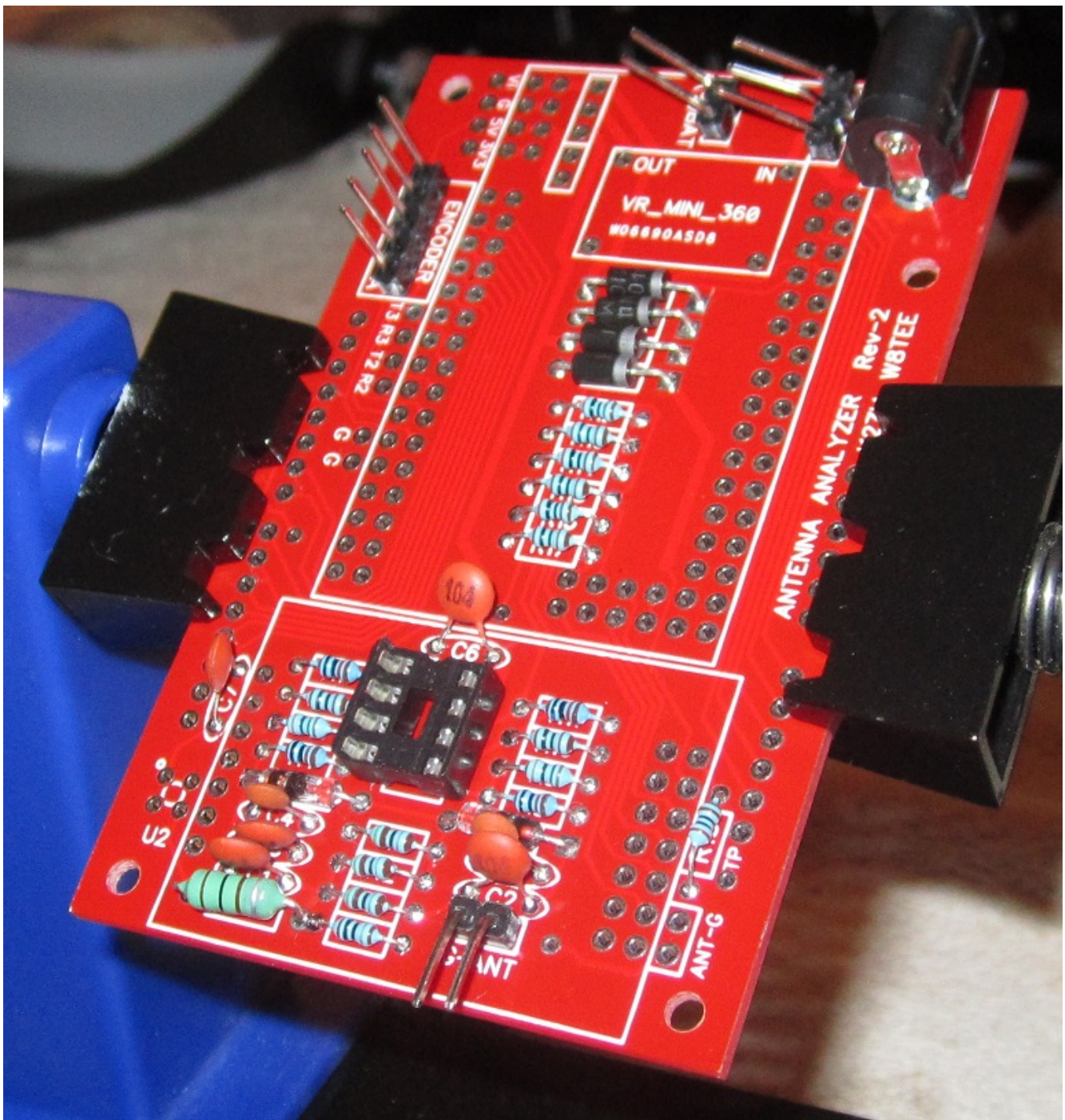
Figure 18. Bend the pins for the antenna, battery, and SW1 connections.

Adjusting the Buck Converter

The AA uses a buck converter to change the 9V from the battery/wall wart to 6.6V. Farrukh went with the buck converter rather than the usual 780X regulator you may have used in the past. The reason is because the converter is almost 75% more efficient than the older style, plus is adjustable. The voltage is adjusted with the small Phillips head screw that can be seen in Figure 19.

If you have a breadboard, you can use old resistor leads for the four pins used for the input and output of the converter. Clip the resistor leads to about a 0.5" length and insert them in your breadboard so

they line up with the holes on the four corners of the buck converter. Slide the converter over the leads and solder into place and trim the leads on the top. The bottom leads will eventually be used to mount the converter on the PCB.

Now hook up a 9V source to the input pins, paying attention to the polarity. Attach the DMM leads to the converter's output terminals. Now turn the Phillips head screw until you see 6.6V on the meter. Try to get as close to 6.6V as possible, although it is not easy to get exactly 6.6V because the adjustment is fairly sensitive. A voltage between 6.3V and 6.9V should be fine.
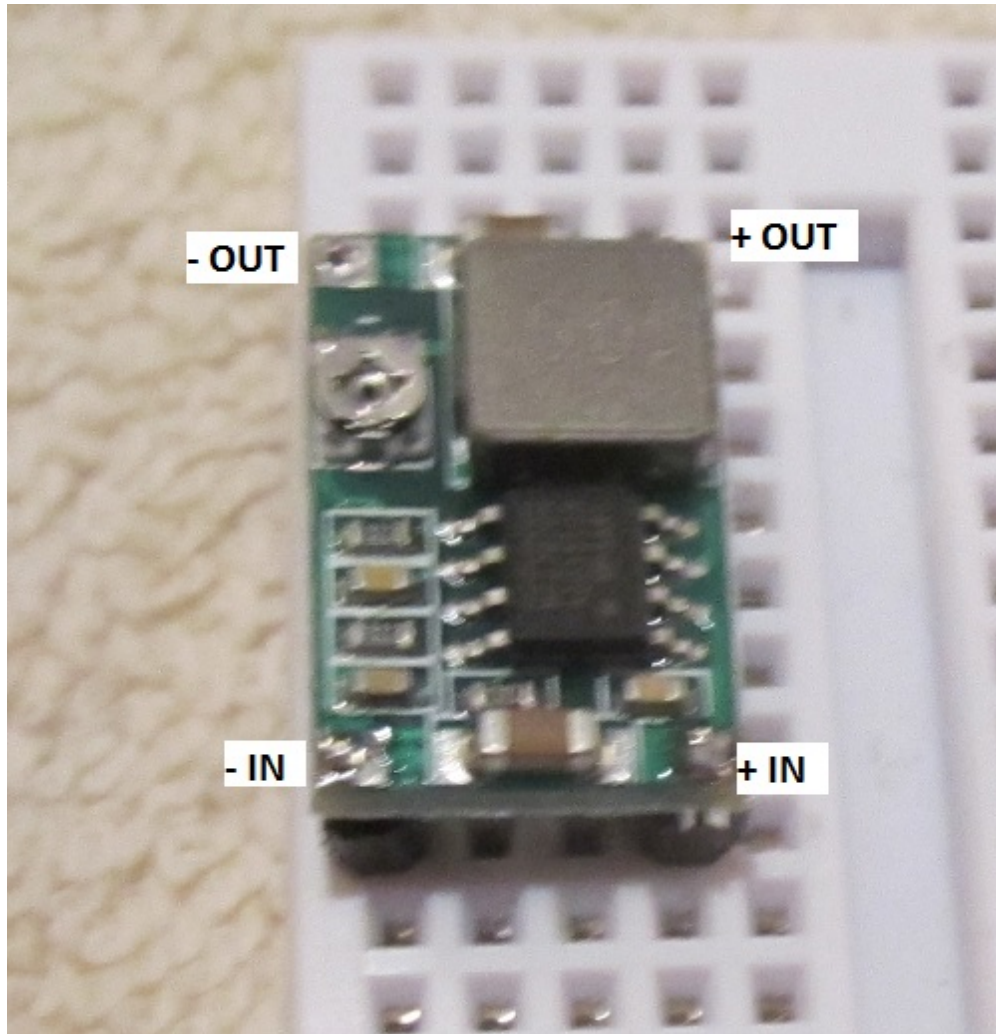

Figure 19. Adjusting the buck converter.

CAUTION: The Phillips head screw on the buck converter shown in Figure 19 is ***very*** fragile and can snap off easily. I recommend a jeweler's screw driver, like that shown in Figure 20. Most of the home improvement stores sell these if you don't already have a set. Just remember not to force the screw...gently does it.

Figure 20. Jeweler's screw driver set.

AD9850 Header Sockets

The AD9850 chip is responsible for generating the frequency signals for testing the SWR. It is a 14 pin
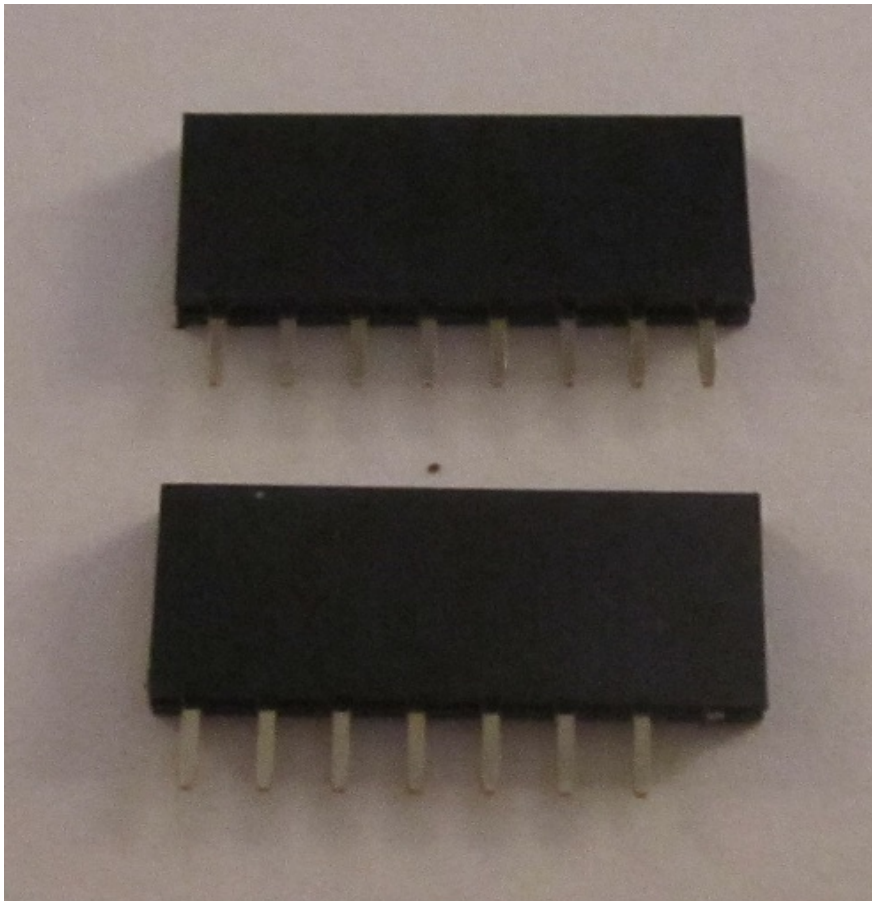
Figure 21. 8 and 7 pin header sockets

chip. At the time the manual was written, the 7 pin header sockets are somewhere between Ohio and China. Rather than hold things up, I used two 8-pin header sockets. If you clip off one of the end pins, the header will still fit on the PCB even though one socket hole will not be used. Hey...any port in a storm. You can see a modified header in Figure 21 with the unmodified header above it. I simply used

my clippers to cut off the right-most lead, yielding a 7 pin socket. Position the sockets so their clipped lead is towards the center of the PCB, as shown in Figure 22. Placing the modified headers in the position makes the 7 active leads on the PCB align with the 7 "good pins" on the header socket. (The photo was taken when I stupidly tried to save six cents worth of I/O pins, mounting only the pins that are used or were needed for stability. Bad idea...fill all of the I/O pin holes.)

Also note in Figure 14 that there are two rows of sockets with 7 holes on the bottom-left side of the PCB and a single row of socket holes near the top-left side of the board. Failure to use the correct top row will mean the AD9850 will not fit in the new socket headers.

When I mounted the modified sockets, I actually fixed the AD9850 module in the two modified socket and placed the socket headers/AD9850 board into the proper holes on the PCB. This way, you can't select the wrong mounting holes. I then flipped the PCB with the AD9850 module over and soldered the socket headers into place.
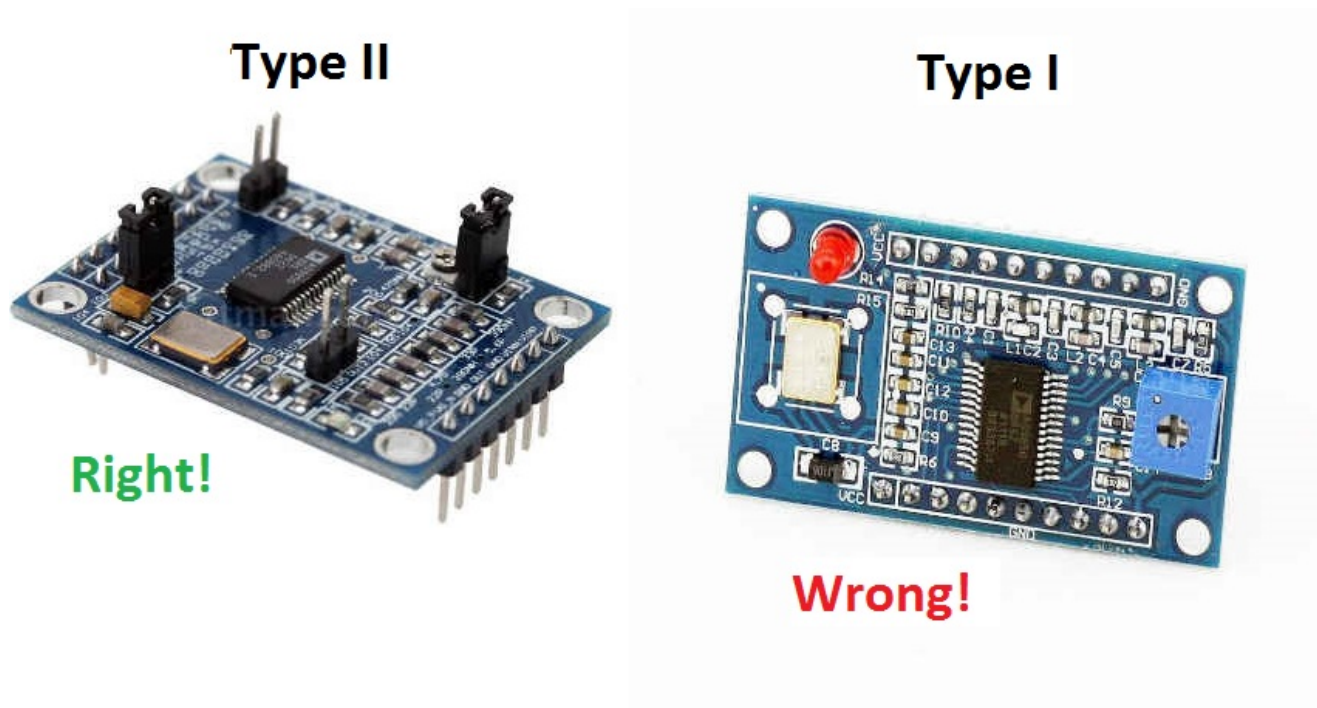

Figure 22. Two different AD9850 boards.

Figure 22 shows that there are two different flavors of the AD9850 modules. The Type II module is what is needed and has a double row of pins on the left edge in Figure 22. The Type I module does not have a double row of pins. Despite eBay vendors showing a Type II module, some vendors don't seem to know the difference and send a Type I, which won't fit the PCB. Send a picture of what you want and always ask if they ship a Type II module and if they don't know what you're asking, find another vendor.

Seating the LM358 Chip

Obviously, you still need to seat the LM358 chip in its socket below the AD9850 board. However, I would prefer that you wait to seat the LM385 until after you run a voltage check. If something was installed incorrectly and the voltage is substantially wrong, you run the risk of "bricking" (i.e., burning a chip into a lump of silicon) one or more chips. Come back to this section after you've finished the Miscellaneous Install section.

However, you can do that at this time, if you wish. Make sure the small indentation on the chip is facing towards the center of the PCB. In terms of Figure 18, the indent should be towards the center of the board. Note that a new LM385 chip has its leads "fanned" outward slightly, so you need to work the leads into the socket. I find that if I place the four leads on the "bottom" of the chip in their socket holes and push sideways slightly, the leads will flex enough that I can slide the "top" 4 leads into the socket. A firm push then seats the chip in the socket.

Check to make sure none of the chip leads "folded" under the chip during the seating process. It happens and it's difficult to see if you're not looking for it. When you place the AD9850 in its socket, there should be clearance between the AD9850 and the LM358 chip...they should not touch.

Miscellaneous Parts Installation

The remainder of the build varies according to your preferences. For example, you may wish to use a different case than the one supplied. Some of you may expect to use the AA "in the field" more often instead of using it in the in-house shack. As such, you may select to use some kind of lithium-ion (rechargeable) 9V battery pack instead of a standard 9V battery. The display does take a fair amount of power, so a regular 9V battery will only last about 30 minutes. If that's what you expect to use, just make sure you turn the unit off between scans. However, you may wish to use a different battery pack if you plan to make a lot of scans in the field. Because the battery pack size may vary, the placement of the power connector for the external wall wart may be positioned differently. I purposely made the case selection bigger than required so that you'd have some wiggle room for such differences.

Personally, I don't expect to be doing extensive outdoor measurements. Because of that, I plan to use a standard 9V battery connected via a cheap (Debco, 4 for $1.00) "snap connector" for the battery for those times when I do expect to be making field adjustments. My choice may not be ideal for you, especially if you plan extensive SOTA, NPOTA, IOTA, backpacking or similar excursions.

Because of the various options that now become available, I will only make suggestions as to how I'm going to do things. Feel free to do whatever you think best.

Enclosure

The enclosure I used comes with 4 screws that attach at the corners, plus a rubber casket that fits inside a groove molded into the lid of the case. See Figure 23. The gasket material is made of white rubber

Figure 23. Enclosure lid with gasket.

and it is pressure-fit into the groove cut into the lid of the enclosure. Figure 23 shows the gasket pushed into place in the lid, plus some of the leftover gasket material. The gasket does *NOT* make the enclosure water tight (after all, there will be holes for the switch and encoder), but does help to keep moisture out.

Gently work the gasket into place using a small flat blade screwdriver. The gasket material tears easily, so take your time. You will likely have some gasket material left over, so trim off the excess. Try to fit the gasket so it forms a solid ring around the lid.

Power Connections

The power connector transfers power from the wall wart to the rest of the AA. I placed mine in the lower right corner of the enclosure, as seen in Figure 24. Once I had the location selected, I took the

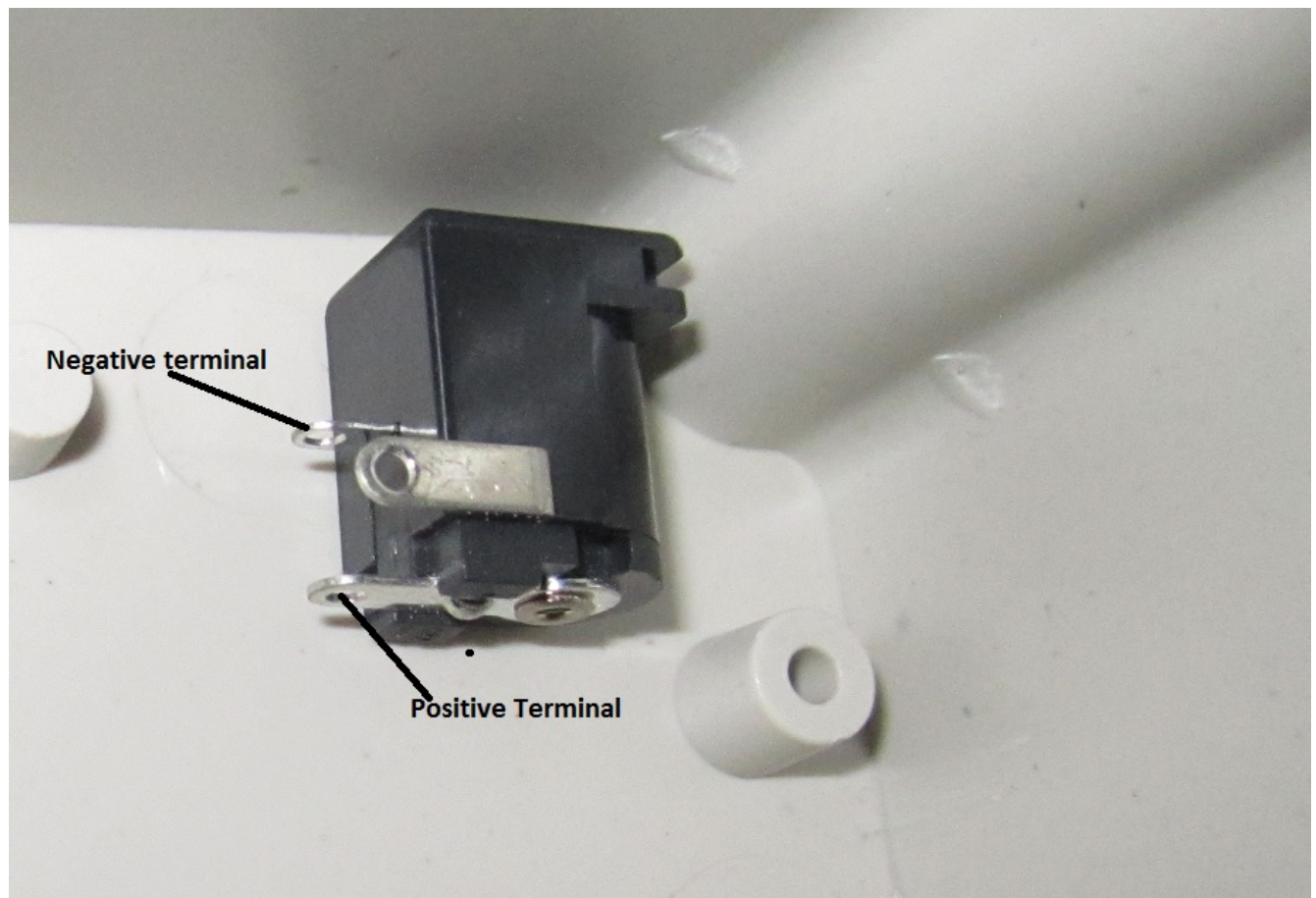*Figure 24. Wall wart connector placement.*

connector and moved it outside the case. I place the enclosure on its side and place the connector on the outside of the case, but opposite where I wanted it located on the inside of the case. I then made a small pencil mark on the case where the top of the connector was located, moving it slightly upward to adjust for the thickness of the plastic bottom. See Figure 25.
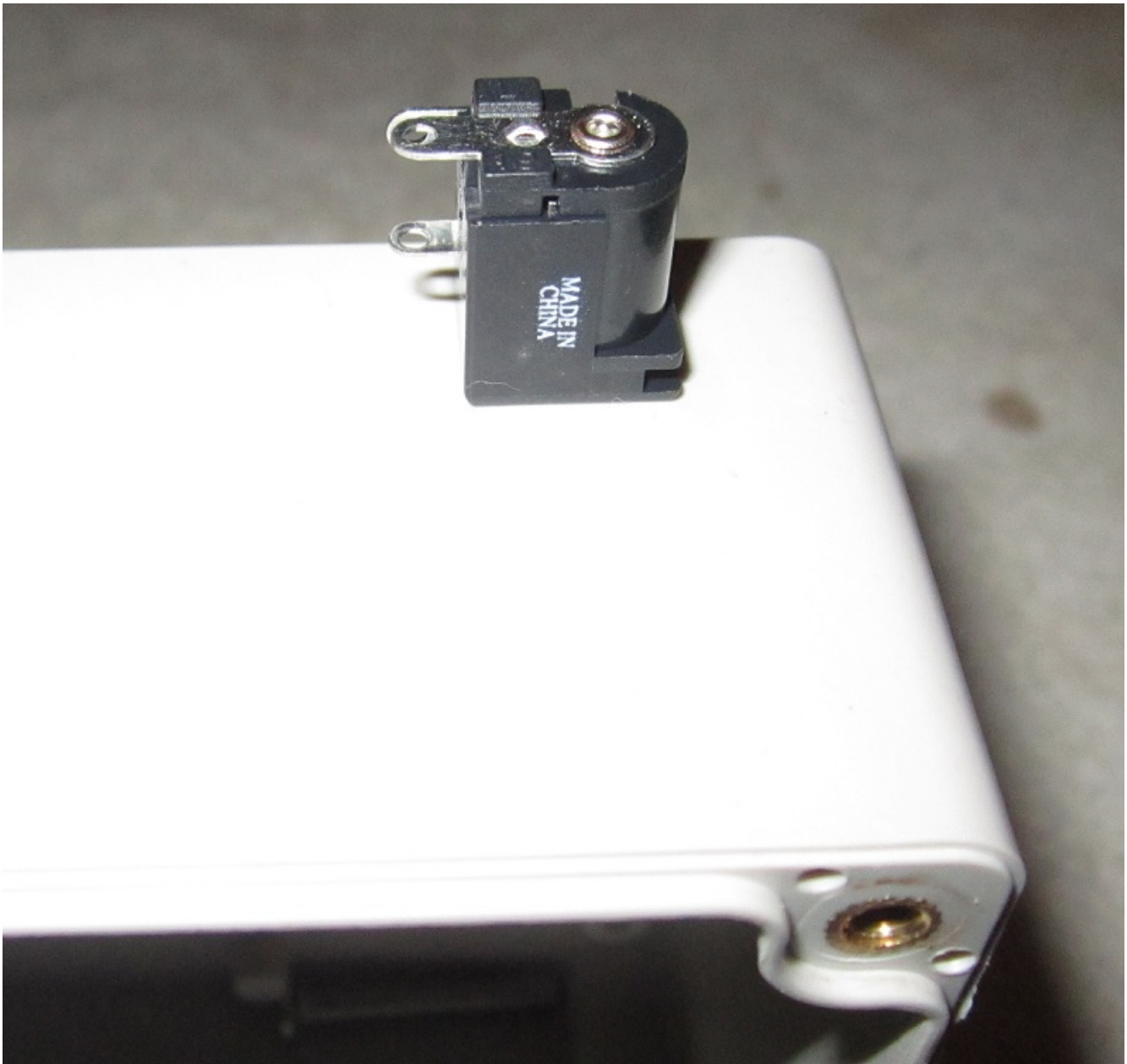
Figure 25. Figuring out where to drill hole for wall wart connection.

After figuring out where the hole for the wall wart plug must go, use a 5/16" drill bit to drill a hole through the case for the connector. Apply a steady, even, pressure as you drill the hole while firmly holding the case. Failure to do hold the case will result in a case that spins just slightly slower than a proton in the Large Hadron Collider.

Now connect two wires to the positive and negative lugs on the connector. (See Figure 24 for lug ID.) If you have red and black wires, so much the better (red to positive, black to negative). I usually put heat-shrinkable tubing on such connections, but it's totally unnecessary.

Now glue the connector in place. I've become a pretty big fan of hot glue for gluing things like this. I use a wall wart plug from the outside of the case to hold the connector in place while the glob of hot glue cools. This assures that the plug and connector align after the glue sets. See Figure 26. (You can see a small filament of hot glue in Figure 26, which I snapped off after it sets.) Do whatever works for you.
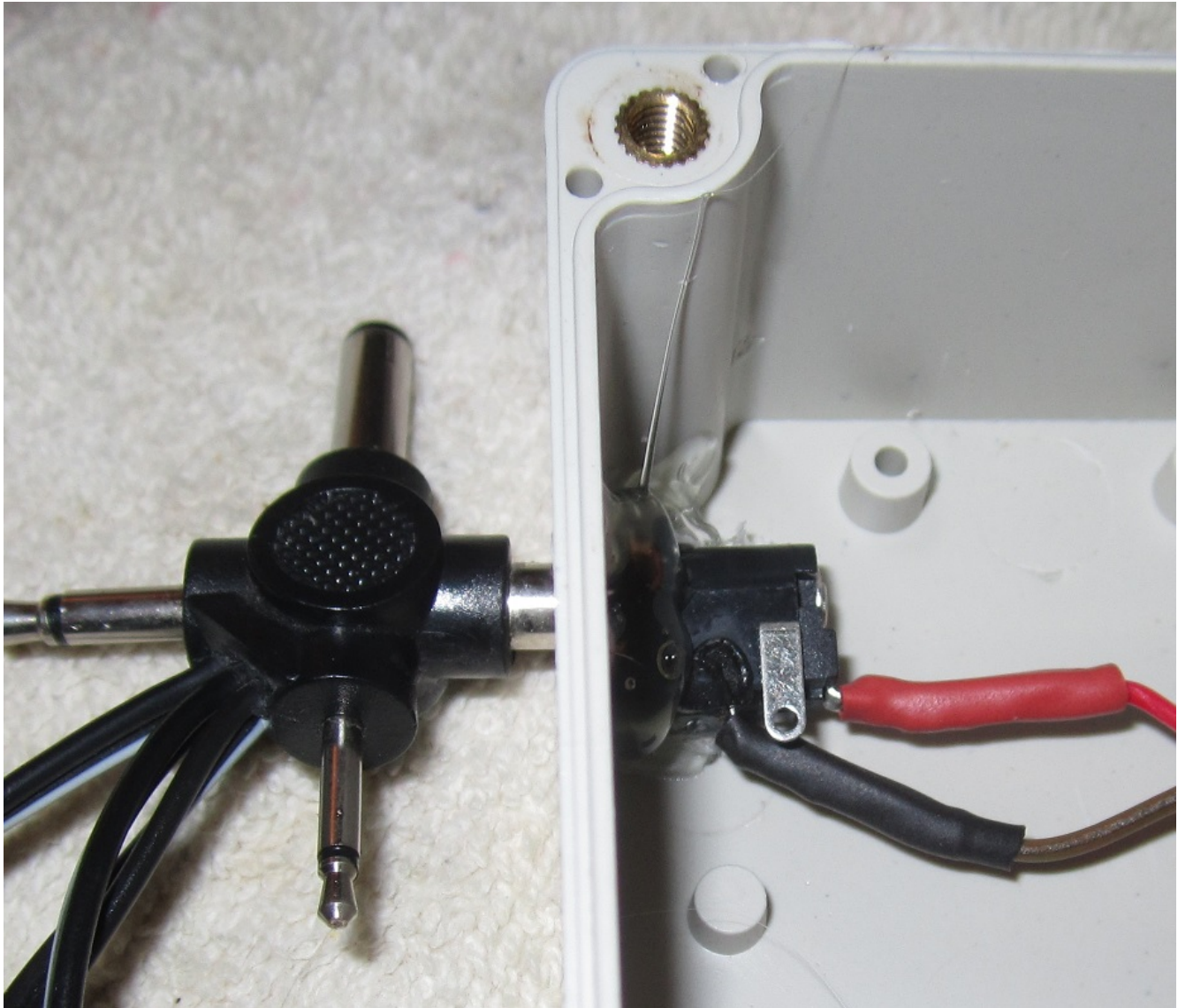
Figure 26. Using the wall wart plug to hold the connector in place while glue cures.

Connect the red (positive) wire from the wall wart connector to the "right outside edge" of SW1, the Single Pole Double Throw (SPDT) switch. In Figure 12, you can see the mounting holes for SW1 just to the left of the power connector in the upper-right corner of the PCB. After that picture was taken, I decided not to use the onboard power connector, as it limited my choices when it came to mounting the display in the enclosure. The three pins that were placed into the mounting holes for SW1 are connected to the switch shown in Figure 28. The "right outside edge" pin of SW1 is the bottom-most pin of SW1 in Figure 12 and is closest to the "IN" mounting hole of the min-386 voltage regulator.

Connect the black (negative) lead from the wall wart connector to the negative mounting hole for the power connector on the PCB. See Figure 27. The board shown in the figure is the same board as shown in Figure 12, so the mounting holes were soldered earlier when I mounted the power connector on the PCB. In hindsight, I wish I hadn't done that, but this manual has taken a lot more time than I expected so you're gonna have to pretend the power connector is not there. Even if it is, you can still solder the negative lead (i.e., the black wire in Figure 26) from the "new" power connector that is glued to the case to the point indicated in Figure 27.
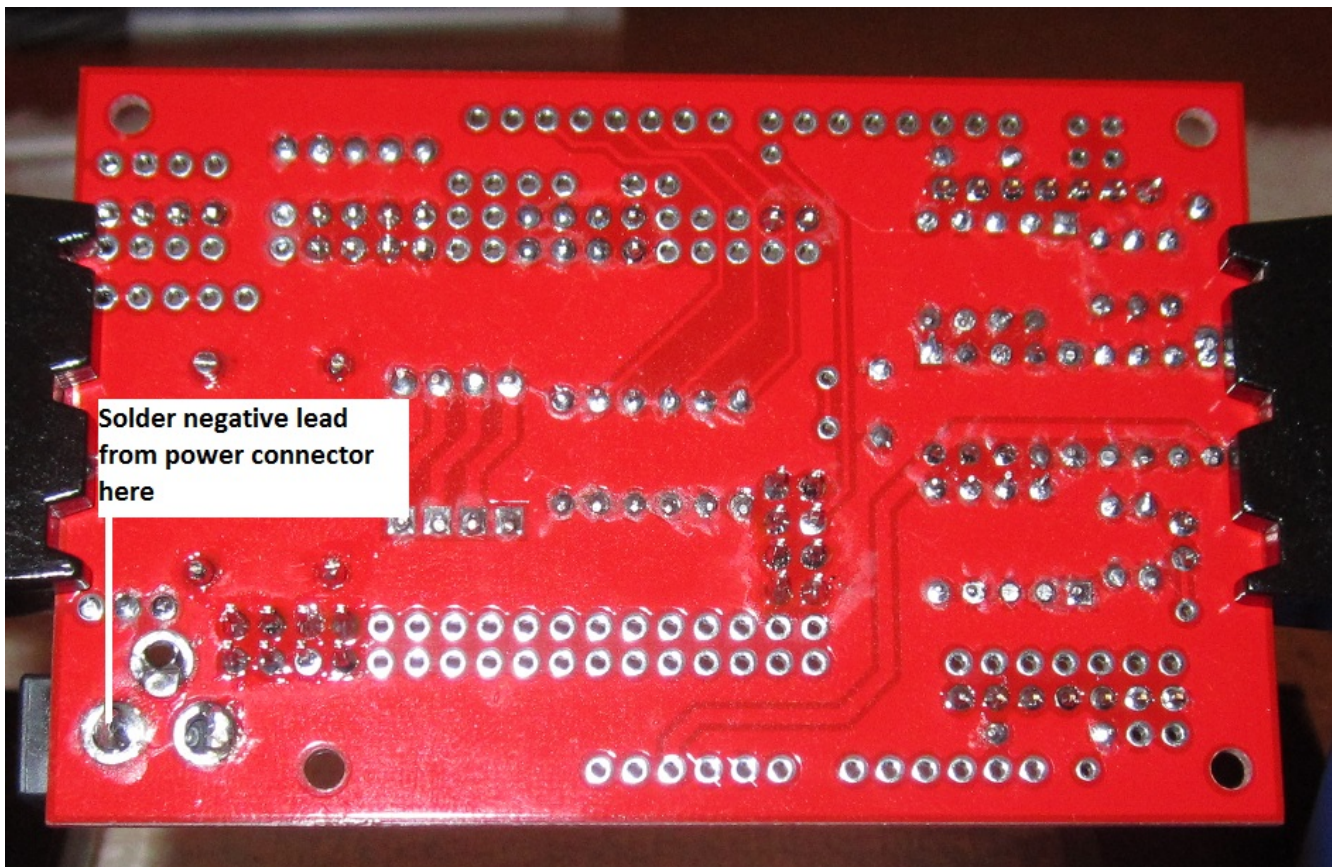
Figure 27. Soldering the negative (black) lead from the case-mounted power connector seen in Figure 26

SW1 is an ON-OFF-ON SPDT switch, so when the switch toggle in in the center position, no power is applied to the AA. No power-on indicator is used, since the TFT display is active when power is present. If the display is blank, power is off. Moving the toggle up or down selects either battery or wall-wart power. You can orient the switch as you see fit in terms of what "up" or "down" means with respect to the switch.

When not in use, switch the AA off, especially when using it in the field so as to conserve battery power. I do not know if these types of displays are subject to "pixel etching", but rather than take a chance, switch it off even if powering it with a wall wart.

To positive terminal of wall wart connector

To "edge" pin of SW1

To center pin of SW1
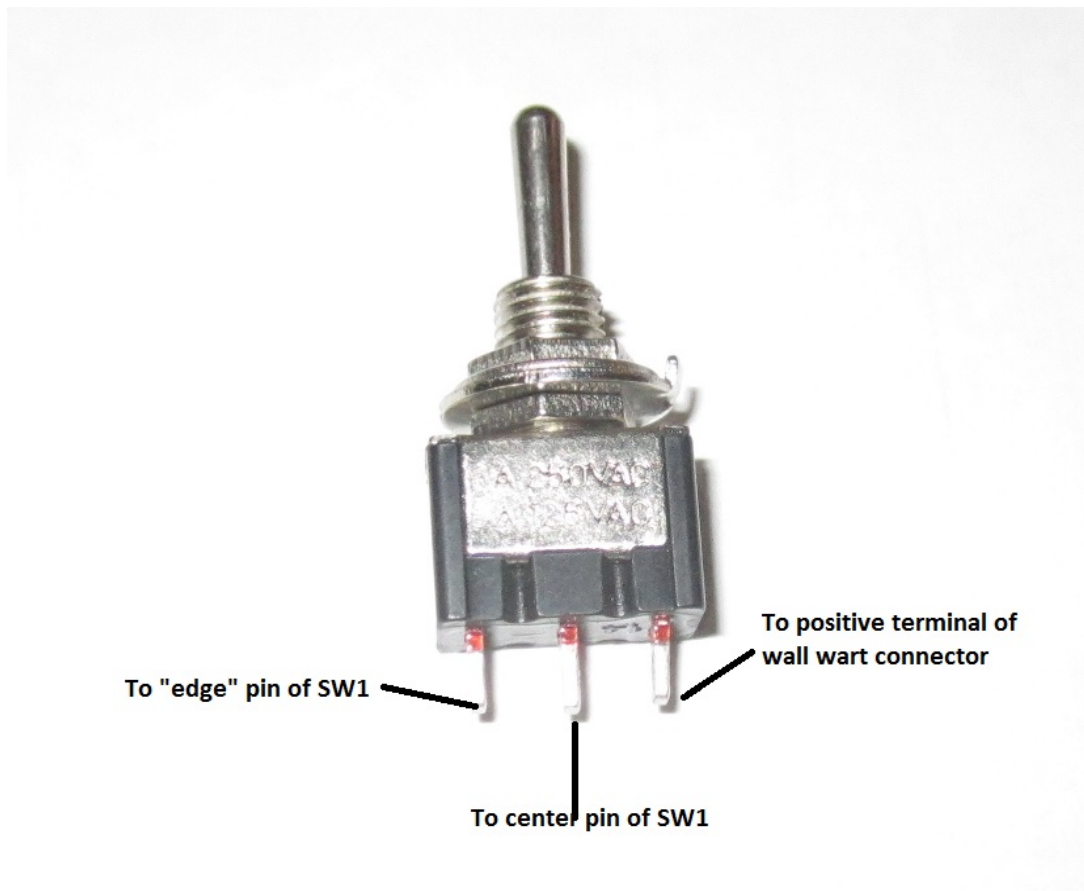
Figure 28. SW1 connections.

Rotary Encoder Connections

The rotary encoder connections are simple. Take 5 of the female-to-female Dupont wire connectors and slide them onto the 5 pins you soldered on the PCB for the encoder. See Figure 14. Make the following connections using the Dupont wires:

Table 4. Encoder Connections

Description of Pin	From PCB Pin	To Encoder Pin
Ground connection	G	GND
Positive 5V connection	+	+
Switch	S	SW
Data connection	B	DT
Clock connection	A	CLK

That's all there is to it for the encoder wiring.

Mounting the TFT Display Headers

You need two more 8-pin socket headers, one 4-pin socket header, and one 6 pin socket header. (You could use two 6 pin headers.) These are arranged on the ***bottom*** of the PCB as shown in Figure 29. The

Scotch tape is used to hold the headers in place when I flip the board over to solder the pins on the top side of the PCB. Note that

Figure 29. The TFT display socket headers

the 4-pin header at the top has two empty holes to the right on the PCB. These pins are not used on the display so we are not connecting them to the board. (If you have the second 6 pin header, place it where the 4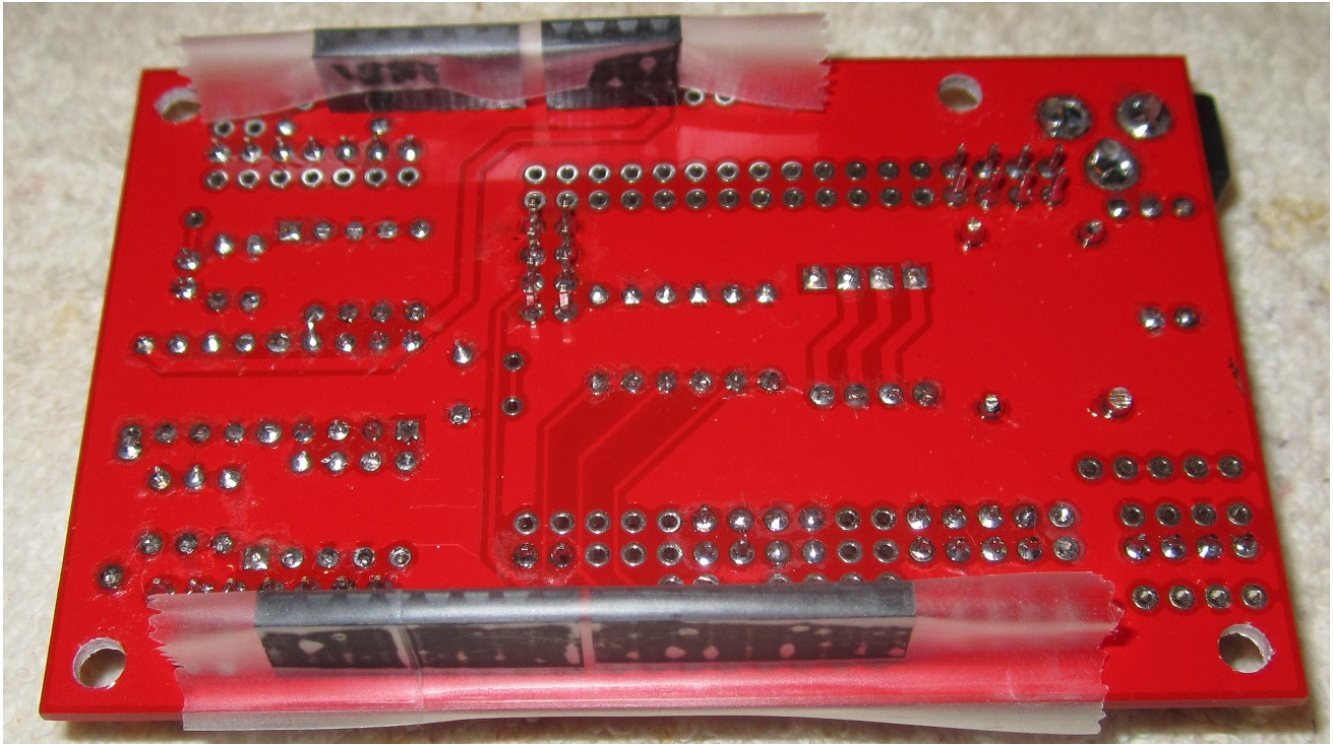 pin header appears in Figure 29.) When you're ready, flip the board over and solder the pins in place. Place a little downward pressure on the board while you solder to make sure the sockets are snug and tight to the PCB surface.

Remove the tape holding the socket headers in place and bend them to make sure they are straight. Now take your TFT display and gently fit it into the socket headers. NOTE: if you mounted the socket headers correctly, two pins on the power-connector side of the 4 pin header will be dangling in the breeze. (You can see these two pins at the bottom of Figure 7.) No problem...it just means you mounted the sockets correctly. All of the remaining pins on the TFT display should be in the socket headers.

Mounting the PCB-Display in the Case

Early in the manual we suggested that you mark the mounting holes for the display. Obviously, you are free to mount things as you see fit, but we are using the format shown in Figure 30. Mark the mounting holes with a pencil. (If you have followed our suggested mounting methods, your case will already have the wall wart connector glue in place in the upper-right corner of the case in Figure 30. We took this picture early before anything was mounted on the PCB.)

NOTE: The TFT display mounts on the "underside" of the PCB, so make sure you have the underside of the PCB (i.e., no silk screening on it) showing when you drill the mounting holes. Failure to do this means you will have to redrill the holes like some other idiot I know did.

The nice thing about this mounting style is that you can remove the SD card or reprogram the Mega2560 without have to undo the bolts since both are accessible from the right side of the board.

Actually, I left the PCB in the case just as you see it in Figure 30. I took a 9/64" drill bit and drilled through the four mounting holes marked in Figure 30. I will use #6-32 brass hardware with 2" long brass bolts. Note that a 9/64" hole is a tight fit for the hardware, which is a good thing. However, the mounting holes are 1/8" on the PCB, so you will end up enlarging the mounting holes in Figure 30 if you use a 9/64" drill bit. That's not a problem and the result is a good, snug fit for the PCB.
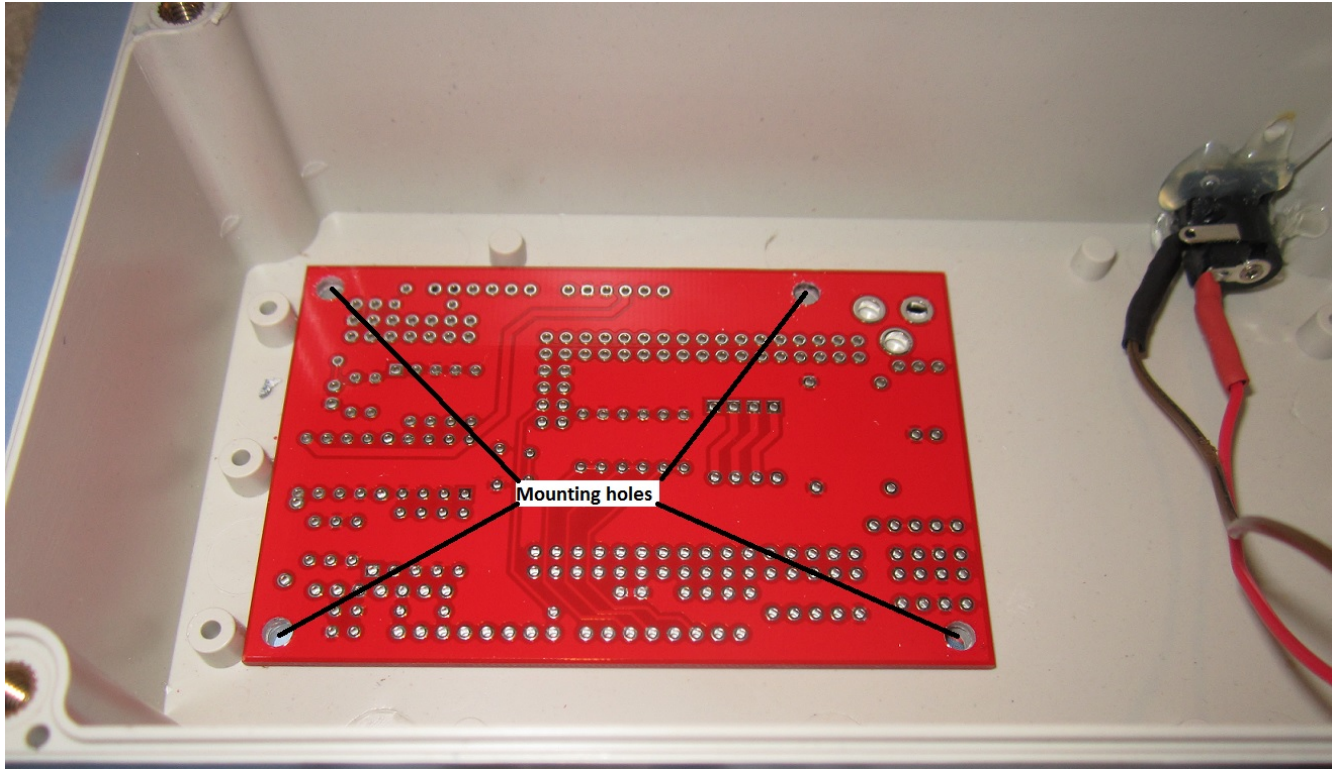
Figure 30. One way to mount the display.

Now thread four nuts on each 2" bolt and tighten to the surface of the case. Now take 4 more nuts and move them down approximately 3/8" from the end of the bolt. Now place the PCB with the TFT display mounted on it on the four bolts. Looking from the side of the case, it should look similar to Figure 31. The second group of nuts supports the PCB in place, but "floats" it above the floor of the
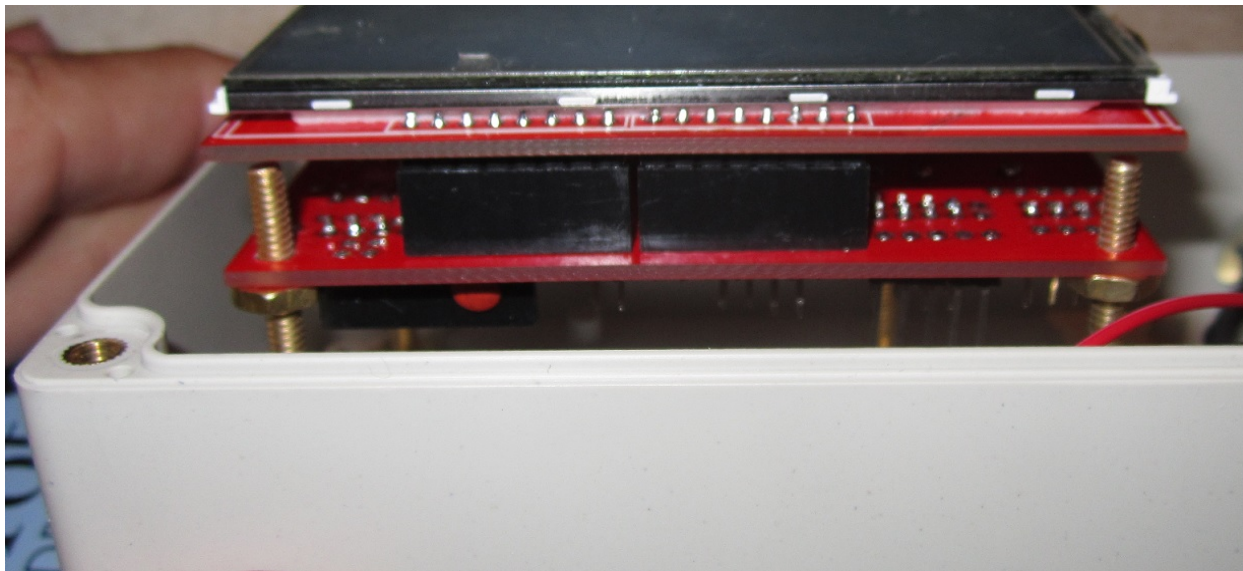
Figure 31. Adjusting mounting nuts for board.

case. Ideally, there should be a little friction between the TFT display and the clear case lid to keep everything rigid. Figure 32 shows what it looks like in the case without the boards in place. The two extra holes towards the back are either for ventilation or were caused by drilling the bolt holes with the board in backwards. (I'm pretty sure it was ventilation.) Once you have the nuts where you want them, place a small dab of glue on each bolt to keep the nut from moving. No nut is necessary on the top of

Figure 32. Mounting bolts for PCB and display.

the board as the case lid friction-fits to keep the display anchored.

Placing the Encoder and Power Switch

Where you place the encoder and the power switch depends upon where you located your display and also the type of battery you plan on using. As mentioned earlier,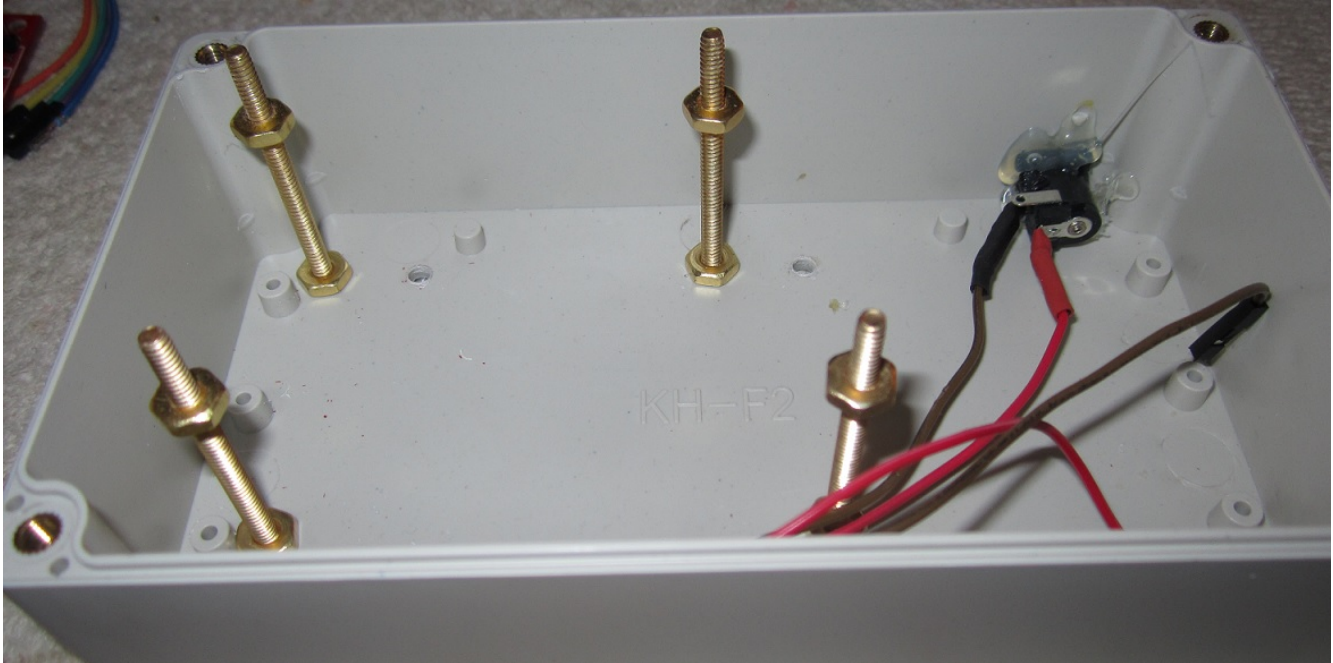 I did not provide a battery connector because you can use any type of 9V power source you wish, from a simple 9V alkaline battery to a 20000mAh Li-ion battery pack. Just keep the voltage between 8 and 12V. If you plan on using the AA in the field a lot, I'd look pretty hard at some form of 9V rechargeable battery packs. For me, a simple 9V battery should be fine, which gives me a lot of options for the encoder and power switch.

One more consideration: where are you going to place the BNC connector for the antenna under test? As you look at the TFT display from the top of the case with the onboard power connector in the upper-right corner, the display is oriented correctly for reading. Given the way I've mounted things in Figure 31, there is a nice gap between the right edge of the display and the right edge of the case, so I'll place the encoder. I'm going to mount my BNC connector on the left edge of the case, away from the wall wart connector and close to the PCB antenna connections. If you are using larger coax connector, you could mount an SO-239 connector to the case, too.

Once you decide on your battery configuration, place the battery pack in the case to check how it fits. That may influence where you place the encoder and power switch. Both items extend into the case, so make sure you have enough room in your layout. I may just place a small eye-hook on the inside of the case and use a heavy-duty rubber band to keep it from rattling around. I really haven't decided on that yet, as I know most of my use will be via a wall wart. Figure 34 shows my positioning.

Drilling the Encoder and Power Switch Mounting Holes

The clear plastic case top is fairly brittle, more so than the gray case body. Once you have selected and marked where you want to place the two controls, select the proper-sized drill bit. I usually select a bit that is just slightly too large to fit through the control's mounting nut. This insures that the control will pass through the case lid, but won't be a loose fit. Do *NOT* use an old, dull, drill bit as it may bind which can fracture the case lid.

With the proper drill bit selected, place the lid with the "lip" facing up so the lid's surface is flat on the drilling surface. I always place an old board on the work bench so I can drill complete through the lid. *Slowly* start drilling the hole with a light downward pressure on the drill. You can't go too slow when drilling these holes. You can, however, go too fast which runs the risk of binding the drill bit and cracking the case. If it takes you ten minutes to drill each hole, good on ya'.

We will attach the wires to each control after mounting the BNC antenna connector.

Mounting the BNC Antenna Connector

You can place the BNC connector anywhere on the case that is convenient for you, as long as it is "out of the way" of the PCB/display board, controls, and power connector. Given the way I have oriented my AA, I elected to mount my on the same side of the case as the power connector, but towards the left side of the case. This places the connector closer to the antenna connections on the PCB.

Select a properly-sized drill bit using the same approach as you did for the encoder and power switch. Solder the Dupont wires to the connector and mount the BNC connector to the case.

Finishing Up

When you are finished drilling the holes, use the Dupont wires to connect the encoder and switch pins to their PCB pins at the points mentioned earlier in the manual. That's it...you're done. Congratulations! You can now read the User's Manual which details how to use the AA.
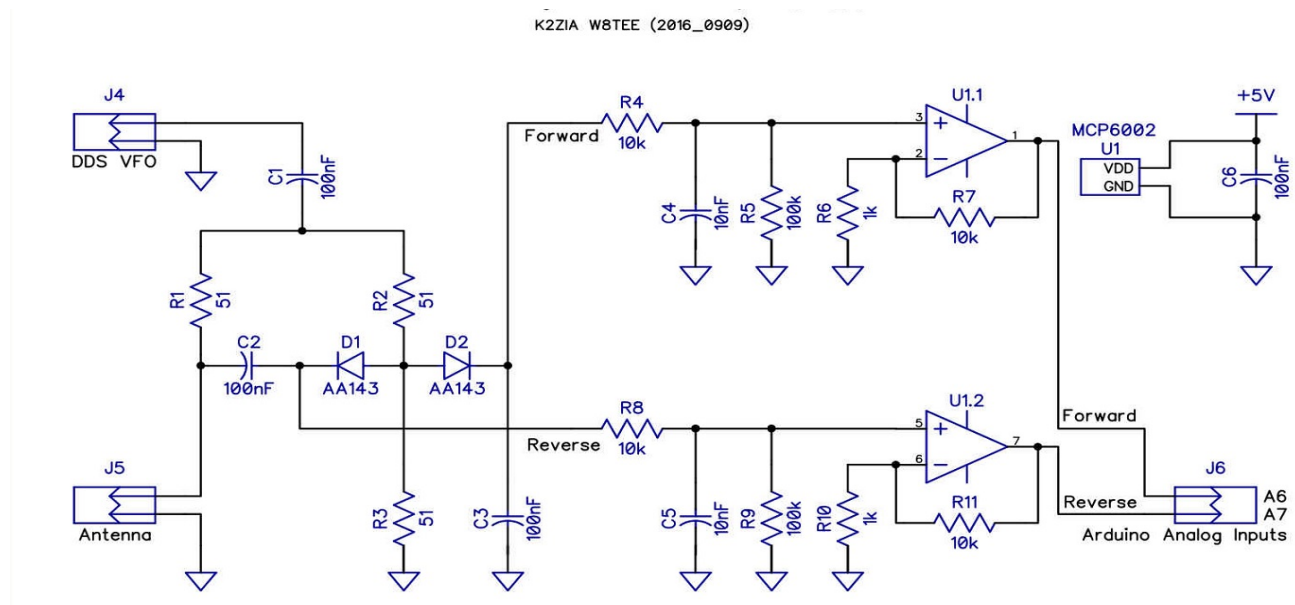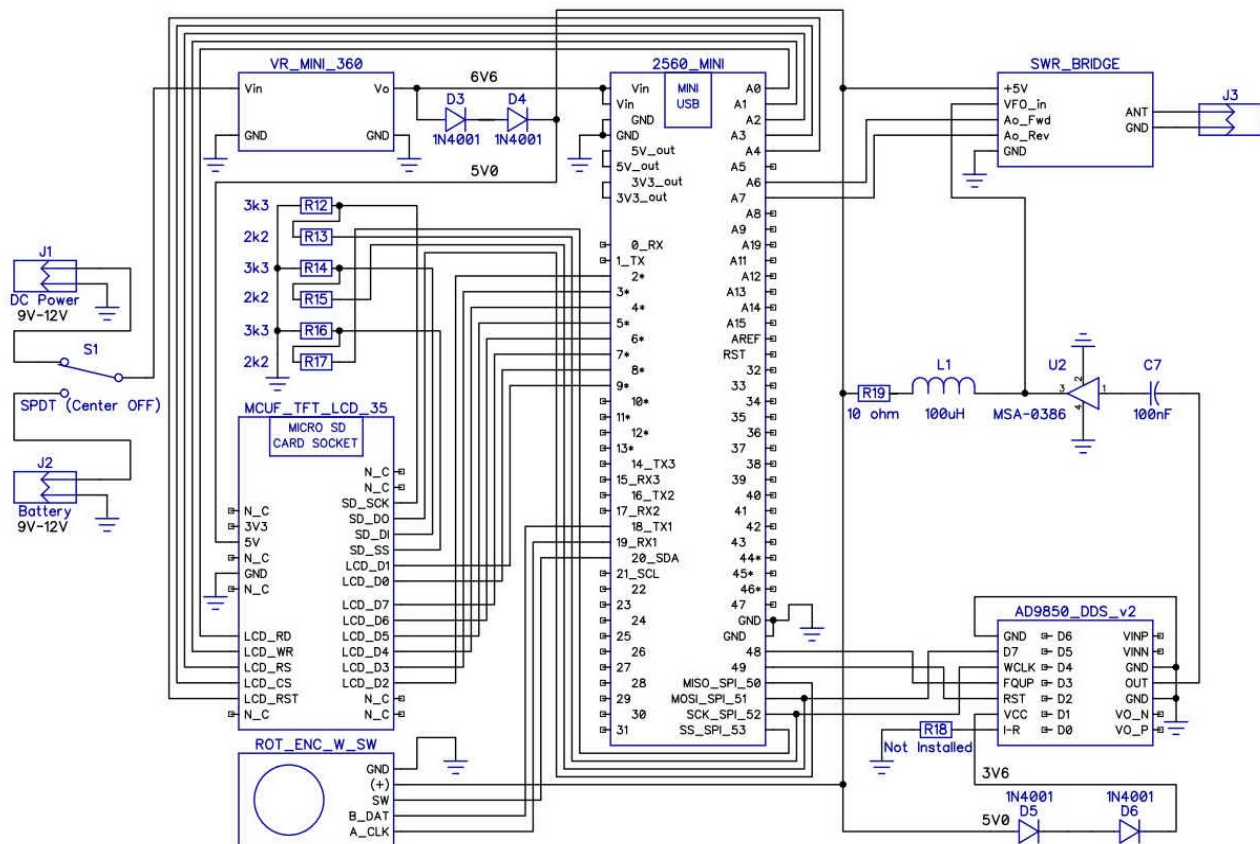

Figure33. Buffer amp for bridge
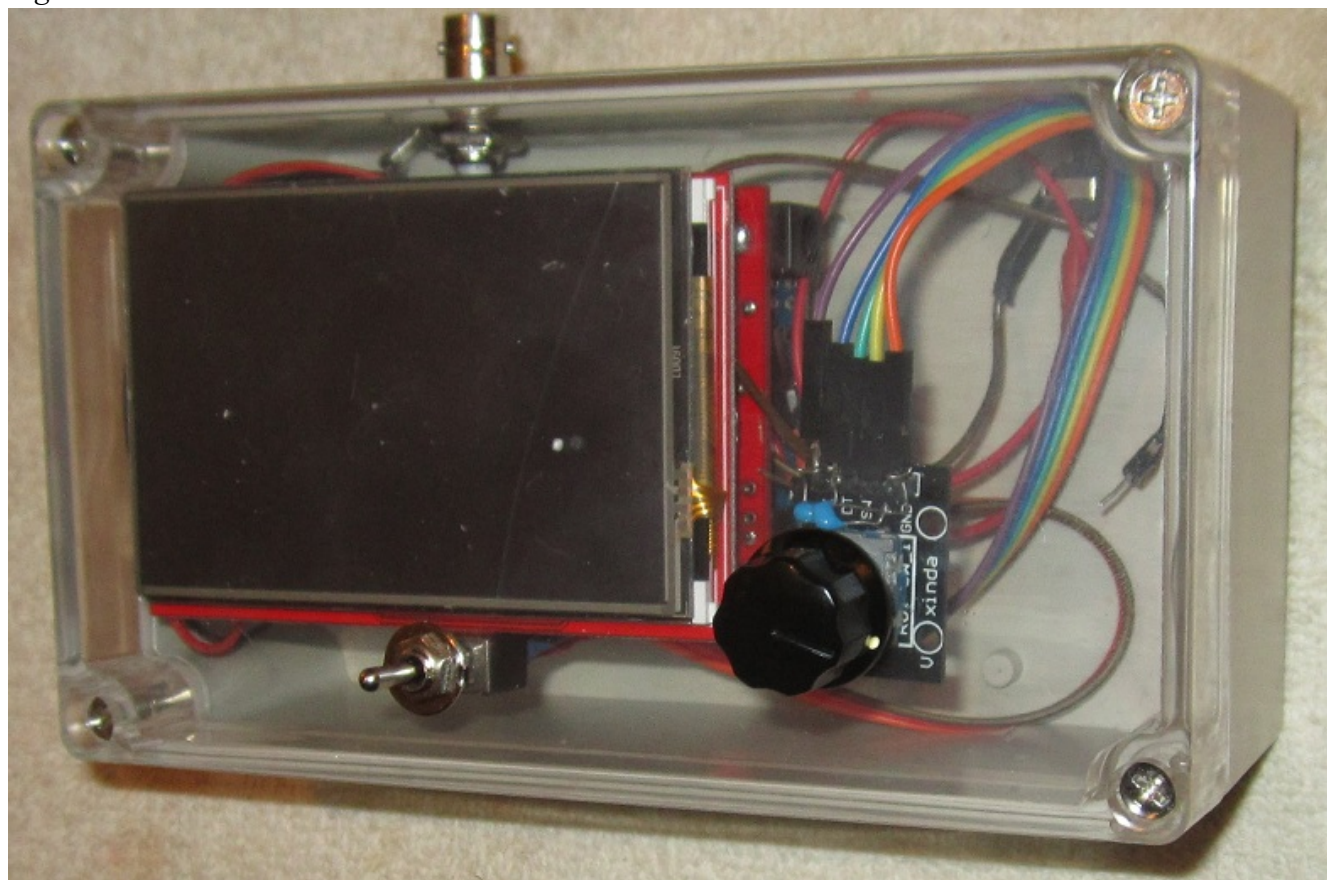
Figure 34. AA Schematic

Figure 35. Finished AA.

Appendix A

Download and Installing the Arduino Software

Arduino Software

A µc without software is about as useful as a bicycle without wheels. Like any other computer, a µc needs program instructions for it to do something useful. Arduino has provided all the software tools within their (free) Integrated Development Environment (IDE) that you need to write program code. The remainder of this article discusses downloading, installing, and testing the software you need.

You need a place on your hard drive for the Arduino compiler and support files. I do *not* like to install it in the default directory (e.g., MyDownloads) because I find it difficult to navigate from the root directory to the Arduino directory. I named my directory Arduino1.6.12 and placed it off the root of the C drive (e.g., [C:\Arduino1.6.12](#)). This is where you will download and install the Arduino compiler and IDE. While you are at it, you might want to create a directory named C:\MARCAntennaAnalyzer, too. You should use this directory to save this manual and the AA source code sketch.

Start your Internet browser and go to:

http://arduino.cc/en/Main/Software

There you will find the Arduino software download choices for Windows, Mac OS X, and Linux. Click on the link that applies to your development environment. The latest Arduino IDE available at the time this is being written is Release 1.6.12. (The title bar in the pictures below says Release 1.6.9, but it's actually 1.6.12 that you are installing. Reshooting the pictures is unnecessary since the process is identical for the new release.) You are asked to select the directory where you wish to extract the files. Browse to your Arduino directory ([C:\Arduino1.6.12](#)). Now download the file and run the installer.

When it finishes installing, look inside the Arduino directory you just created and double-click on the arduino.exe file. This should cause the IDE to start. After the IDE loads into memory, your screen should look similar to Figure A1. You should have your Arduino Mega2560 Pro Mini board connected to your PC via the appropriate (mini A) USB cable for your
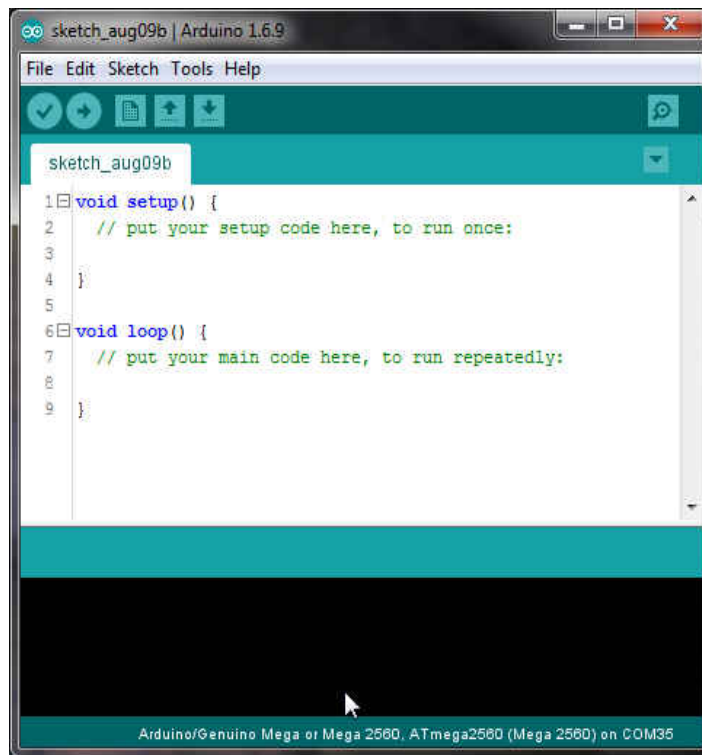
Figure A1. The IDE startup screen.

board. All that remains is to select the proper Arduino board and port. Figure A2 shows how to set the IDE to recognize your board. As you can see, the one IDE supports a large number of the Arduino family. Click on the board that you are using.
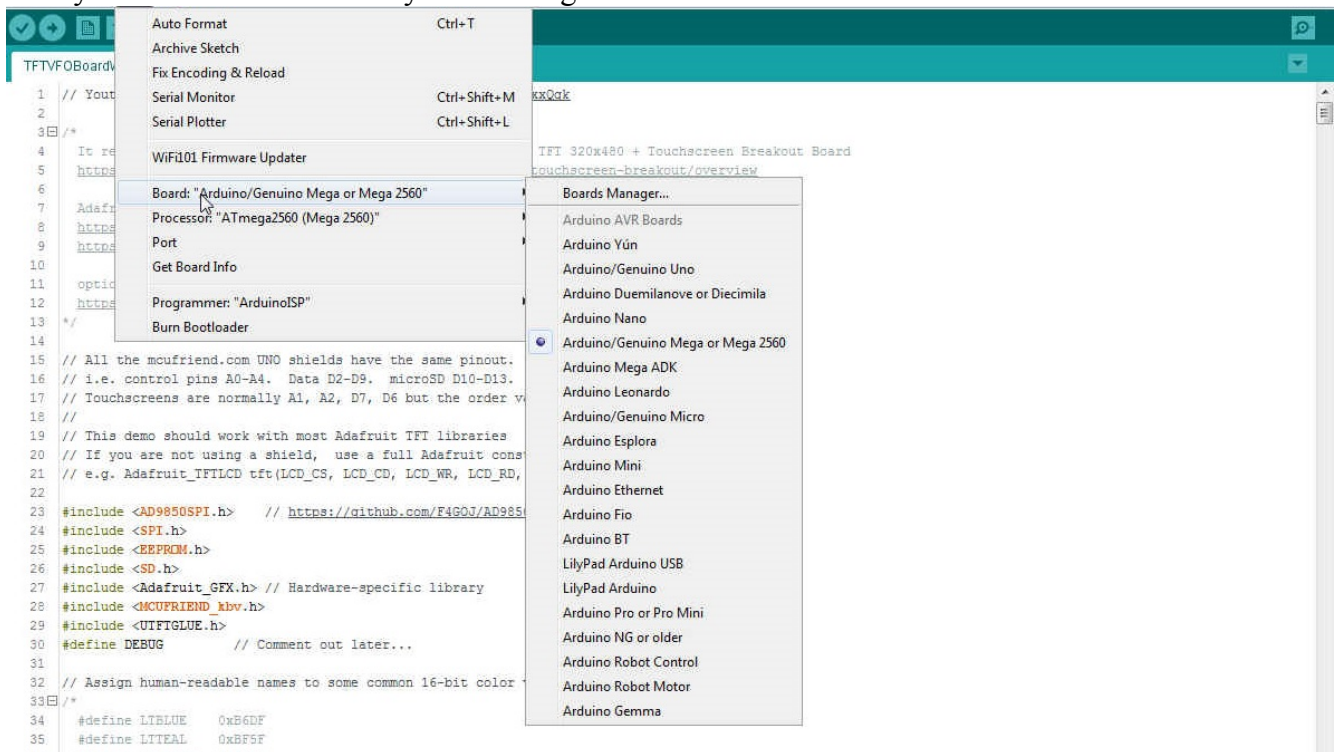
Figure A2. Setting the Arduino board to use the Mega 2560 processor.

Now set the port number for the USB port that is being used to communicate between your Arduino and the PC. This is shown in Figure A3. Sometimes the Windows environment does not find the proper port. This is usually because the device driver for the port isn't found. If this happens, go to your installation directory and into the *drivers* subdirectory (e.g., C:\Arduino1.6.12\drivers) and run the driver installation program that's appropriate for your system. (Figure A4 shows the *drivers* subdirectory contents.)
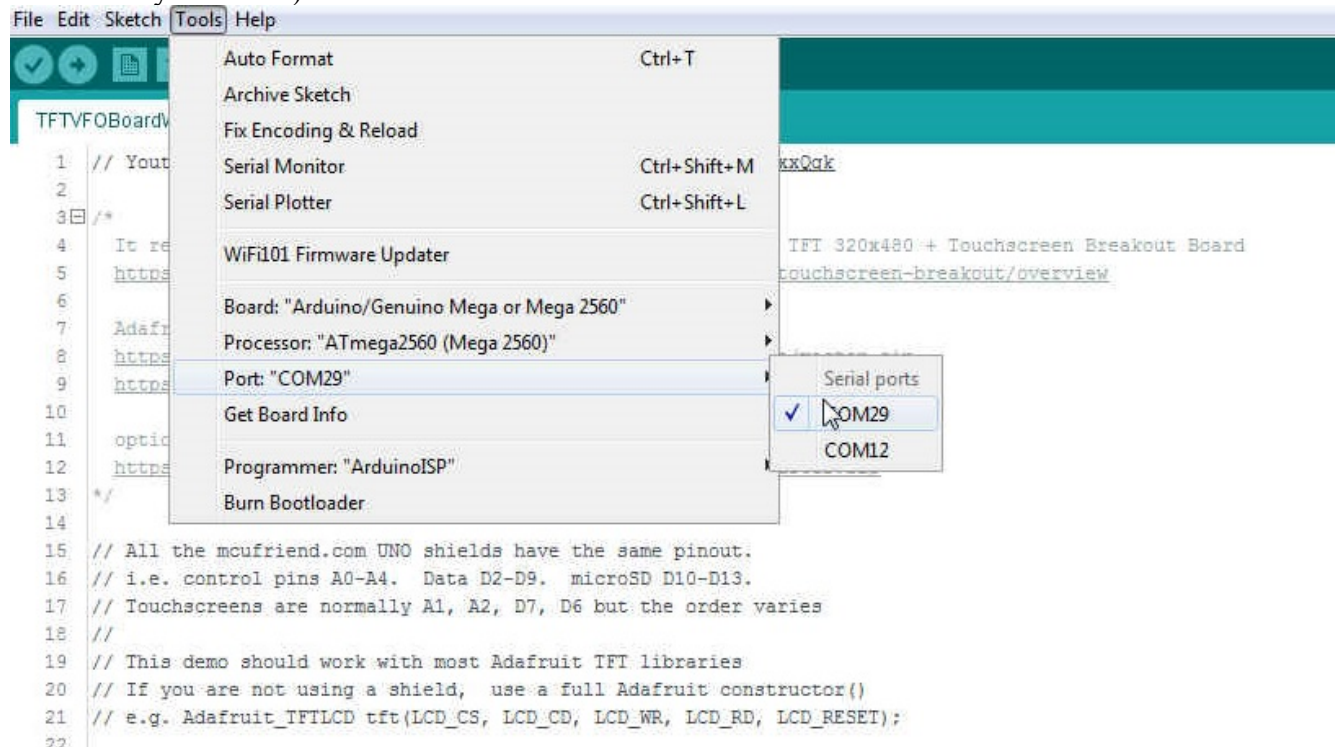
Figure A3. Setting the communications port for the Arduino.

Figure A4. The drivers subdirectory

Once the driver is installed, the program should now find your Arduino COM port.

The Integrated Development Environment (IDE)

The easiest way to test that everything installed correctly is to run a program. Your IDE has a program called Blink. You can find it using the menu sequence: File → Examples → 01. Basics → Blink. You can see this sequence in Figure A5. Once you click on Blink, the IDE loads it into the Source Code window.
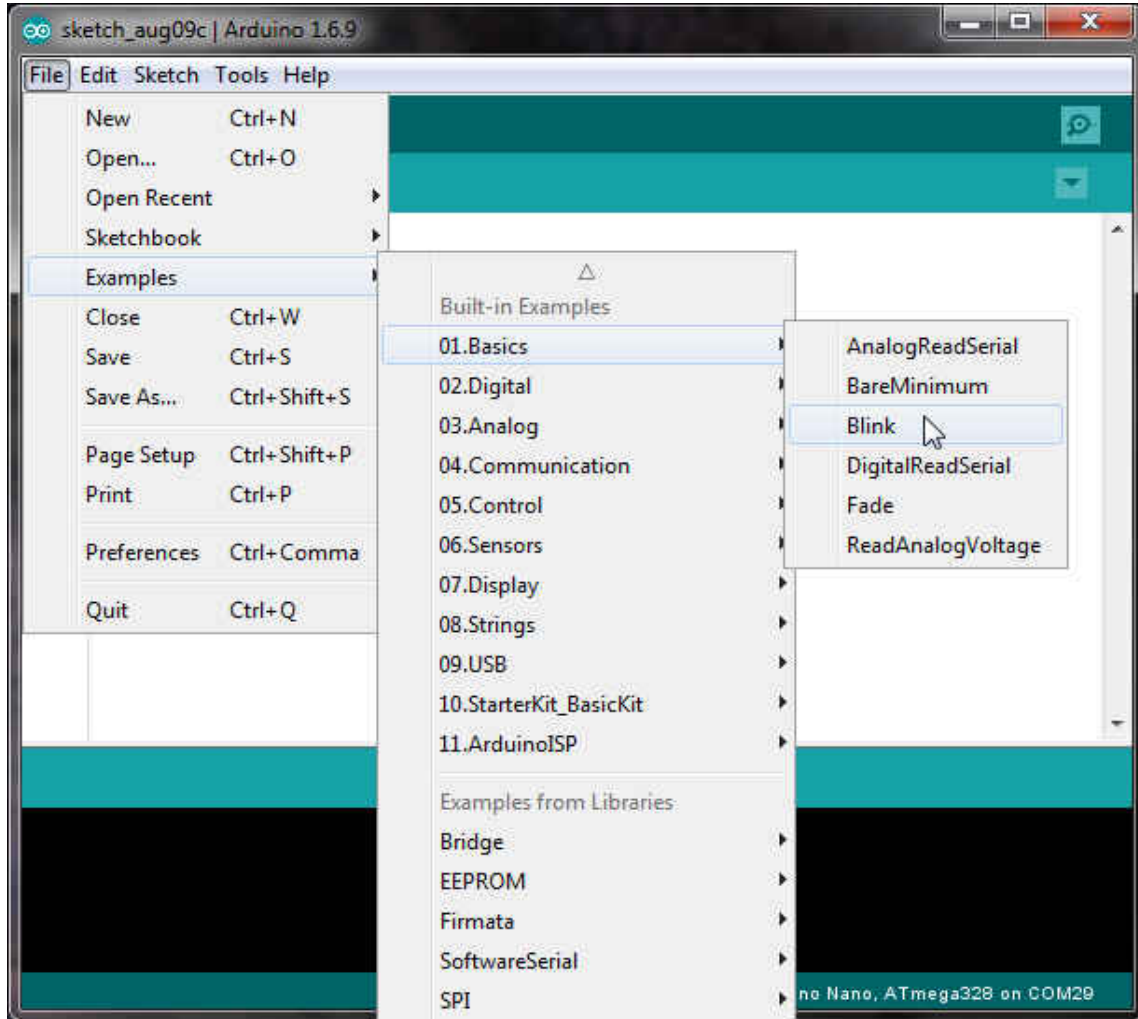
Figure A5. Loading the Blink program.

Once the Blink program is loaded, the IDE looks similar to Figure A6. I've marked some of the more important elements of the IDE in the figure. Starting near the top is the Serial Monitor icon. You click on this to see any print statements you might be using in the program. We'll show an example of such statements in a moment.

The large white space is where you will write your program source code. Program source code consists of English-like instructions that tell the compiler what code to generate. Because you already loaded the Blink program, you can see the Blink source code in Figure A6.

The Compile Icon is exactly that: It compiles your program. It does not, however, link all the parts of the program together to form an executable program. Using the Compile Icon is a quick way to see if your have the code syntax correct.
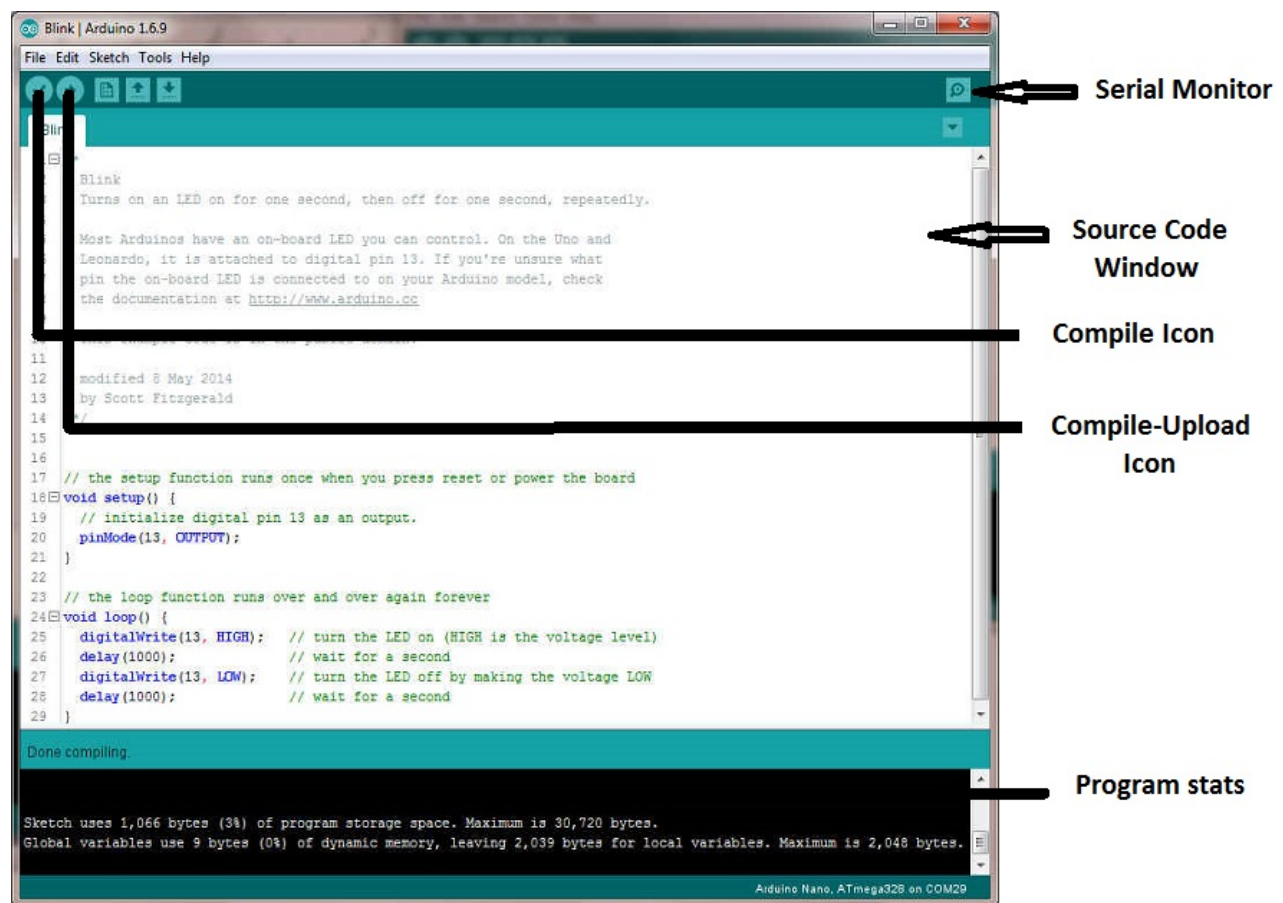
Figure A6. The IDE.

The Compile-Upload icon not only compiles your program, but it links it into an executable program and then transfers the program code to the Arduino via the USB cable. In other words, your PC is used to write, test, compile, and debug your program, but the code actually runs on the Arduino. The Arduino has a small program, called a bootloader, that manages all of the communications between your PC and the Arduino. You don't need to worry about it. (If you compile a program on a Nano, it will tell you that you have about 30K of program space even though it's a 32K memory bank. The "missing" 2K is the bootloader. The picture was taken while using a Nano rather than a Mega 2560.)

The Program stats window tells you how much flash and SRAM memory your program is using. The window is also used to display error messages if the IDE detects something wrong with your program.

Your First Program

Let's look at the Blink program and make a couple of simple changes to it. The code is reproduced in Listing 1, with all of the comments stripped away. Look at the element called *setup()*. The element is

Listing 1. Blink source code.

```
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
```

```
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```
actually called a function. A *function* is a group of one or more program statements designed to perform a specific task. The purpose of the *setup()* function is to define the environment in which the program is to operate. Without getting too deep right now, *pinMode()* is also a function which has been predefined for you to establish how a given Arduino I/O pin is to operate. In this case, we are saying that we want to use pin 13 to output data. However, every Arduino happens to have a small LED available for use on pin 13. In this program, therefore, we are going to use pin 13 to output data. As it turns out, the output data consists of blinking the LED.

One thing that it unique about *setup()* is that it is only called once, and that is when you first apply power to the Arduino or you press its Reset button. Once it has defined its operating environment, its task is complete and it is not recalled.

The *loop()* function, however, is called continuously. If you look at the code, the first statement is a call to the *digitalWrite()* function. It has two function arguments, 13 and HIGH. These two pieces of information (i.e., the pin number and its state) are needed by the *digitalWrite()* function to perform its task. The function's task is to set the state of pin 13 to HIGH. This has the effect of putting 5V on pin 13, which has the effect of turning the LED on. The next program statement, the *delay()* function call, has a number between its parentheses. This number (i.e., 1000) is called a *function argument* which is information that is passed to the *delay()* function code which it needs to complete its task. In this case, we are telling the program to delay executing the next program statement for 1000 milliseconds, or one second.

After one second, another *digitalWrite()* function call is made, only this time it sets the state of pin 13 to LOW. This turns off the LED tied to pin 13. Once the LED is turned off, the program again calls delay() with the same 1000 millisecond delay. So, collectively, the four program statements in *loop()* are designed to turn the LED on and off with a one second interval. That almost sounds like blinking the LED, right?

Now here's the cool part. After the last *delay()* is finished, the program loops back up to the top of the *loop()* function statements and re-executes the first program statement, *digitalWrite(13, HIGH)*. (The semicolon at the end of the line marks the end of a program statement.) Once the LED is turned on, *delay()* keeps it one for once second and the third program statement is again executed.

Now press the Compile-Upload icon and after a few seconds you will see a message saying the upload is done. If you look at your Arduino, it's now sitting there blinking its LED for you.

Your program continues to repeat this sequence until: 1) you turn off the power, 2) you press the reset button (which simply restarts the program), or 3) there is a component failure. If your program is performing as described here, you have successfully installed the IDE and compiled your first program.

A Simple Modification

Let's add two lines to make this "your" program. Move the arrow cursor into the Source Code window. The cursor will change from an arrow to an I-bar cursor. Now type in the new lines shown in Listing 2.

Listing 2. Blink modifications.

```
void setup() {
  // initialize digital pin 13 as an output.
  Serial.begin(9600);
```

```
  Serial.print("This is Jack, W8TEE");
  pinMode(13, OUTPUT);
}
```

Obviously, you should type in your name and call. The first line says we want to use the IDE's Serial object to talk to your PC using a 9600 baud rate. The Serial object has a function embedded within itself named *begin()*, which expects you to supply the correct baud rate as its function argument. The second line simply sends a message to your PC at 9600 baud over the Serial communication link (i.e., your USB cable). If you click on the Serial Monitor icon (see Figure A6), you will see the message displayed in the Serial monitor dialog box. At the bottom of the box are other baud rates that are supported by the IDE. The *begin()* baud rate and the rate shown at the bottom of the box must match. If they don't, your PC will blow up! Naw...just kidding. Actually, you may not see anything at all or you may get characters that look a lot like Mandarin.

Adding Two New Required Libraries.

Most Arduino programs do not use a TFT display like the AA uses. For that reason, we need to make special software available to the Arduino compiler so it can process commands to the display. The special add-on software is found in three new libraries we must download.

A software library is a lot like a book. If you have a reference book and want to look up a chapter on a specific subject, you open the book to the Table of Contents (TOC), read down the chapter headings until you find the topic you're interested in, then you scan across to see what page number to turn to. Our library works much the same. Suppose there is a function named *begin()*, which is used to set up certain variables to default values. You open the library and scan an ordered list of topics (e.g., the TOC for the library), find *begin()*. However, instead of scanning over and reading a single page number, the library places two numbers after the *begin()* entry. Those numbers might be 1100, 85. The first number says: skip over 1100 bytes of library code to get to the start of the *begin()* code. The second number says to read 85 bytes, which is the number of bytes needed by *begin()* to accomplish its task.

When the compiler gets to the place in your program where you call the *begin()* function, the compiler takes those 85 bytes and sticks them in your program. The benefit of the library is that you didn't have to write the *begin()* code. It comes to you written, debugged, and fully tested. This is a bit of a simplification, but it does reflect the basic process of how a library is used in your programs.

If you installed your Arduino software as we suggested, you should see something like:

C:\Arduino1.6.12\drivers
>**examples**
>**hardware**
>**java**
>**lib**
>**libraries**   **<--- Install new libraries in this subdirectory**
>**reference**
>**tools**
>**tools-builder**
>**arduino.exe**
>**// more files...**

The new library files go under the *libraries* subdirectory. The compiler comes with a dozen or so libraries already installed. We just need to add three new ones to use the TFT display. First, go to the

following URL:

https://github.com/F4GOJ/AD9850SPI/

and click on the Download button and then select the Download Zip option. It will then ask you where to save the ZIP file. Select the *libraries* subdirectory shown above.

Now repeat the process for the following:

https://github.com/adafruit/Adafruit-GFX-Library

https://github.com/prenticedavid/MCUFRIEND_kbv

https://github.com/frodofski/Encoder_Polling

Now, one-by-one, double click on a ZIP file and the Windows Explore program will ask you if you want to Extract Files. Click on that option (look towards the topic of the window to see the option). Repeat for the other two library files. The order that you do them does not matter. It will then extract and unpack all of the files associated with the libraries.

When you double clicked on the AD9850SPI ZIP file, Windows Explore creates a subdirectory named

        C:\Arduino1.6.12\libraries\AD9850SPI-master

You need to rename that subdirectory to:

        C:\Arduino1.6.12\libraries\AD9850SPI

In other words, you got rid of "-master" from the subdirectory name. If you go inside of that subdirectory, you will see the old subdirectory name *AD9850SPI-master*. If you look inside that subdirectory, you will see two directories and a bunch of files:

        C:\Arduino1.6.12\libraries\AD9850SPI\AD9850SPI-master\examples
                                        images
                                        AD9850SPI.cpp
                                        // a bunch more files...

What you need to do is copy every file in the subdirectory to the new AD9850SPI directory. When you are done, your directory should look like:

        C:\Arduino1.6.12\libraries\AD9850SPI\AD9650SPI-master
                          examples
                          images
                          AD9850SPI.cpp
                          // a bunch more files...

Now delete the (now empty) *AD9850SPI-master* subdirectory so it finally looks like:

        C:\Arduino1.6.12\libraries\AD9850SPI\examples
                          images
                          AD9850SPI.cpp
                          // a bunch more files...

As a result of this, and the really important thing, is that the libraries\AD9850SPI direcotry have the AD9850SPI.cpp and AD9850SPI.h immediately below their subdirectory name. If you don't do this correctly, the compiler will give you an error message saying it cannot find the needed file(s).

The exact same process is done for the other two ZIP library files, only using their names. When you are done, you should see three new subdirectories in the *libraries* subdirectory:

> AD9850SPI
> Adafruit_GFX
> MCUFRIEND_kbv
> Encoder_Polling

If you have problems compiling the AA program, I can just about guarantee you it's because you don't have the libraries installed where they need to be installed. So, if you have problems, this is the section you should check first.

NOTE: I have set the number of samples per frequency at 75. This appears at approximately line 48 in the code file:

#define MAXPOINTS 75

This is a tradeoff. The AD9850 does produce a little noise on its sine wave. That wave is propagated to the antenna, and some of that energy is reflected back and used in the SWR calculation. If you set MAXPOINTS to 1, it run very fast, but the plot is "jittery" because of the line noise plus other factors. Even the wind swaying the antenna can have an effect. As you increase the number of sample points, the jitters tend to smooth out because we use the average of those sample points. The more sample points, the smoother the curve, but the longer it takes for a scan. Unlike some AA's that use maybe 10 or 12 samples in the scan, we use 100 different frequencies within the give scan range. Therefore, if you set MAXPOINTS to 75, you are actually reading 7500 data points and doing the math on all those points. Even so, a scan takes only about 5 seconds.

You are free to set MAXPOINTS where you want—higher or lower—but realize the impact those changes have on the result and the time it takes to get that result. While I have not done exhaustive tests, it appears that the sample variance on a scan using 75 points is less than 1% at 40M. I would expect that to about double on 10M, as there is some deterioration in the DDS sine wave as the frequency increases.

The Code

Because this is going to be a commercial product, I had to convince the company that it would pose no threat to give the members the source code for the AA. You are free to modify it, but not to distribute it. I ask that you do your best to insure that the code remains in this group. A stripped down version of this code may appear in an article, provided I can get that code approved for release.

Note: You should NOT cut-and-paste the code from this document into the IDE and try to compile it. The reason is because some text editors replace the ASCII double quote characters for "prettier" double quote marks that "slant in" towards the phase being quoted. Because the Arduino IDE does not know about these special marks, the compiler throws an error message. The nature of the message is not very helpful and you may spend a lot of time trying to figure out the source of the error. Instead, please use the *AAWithInterrupts.ino* file provided for the project.

In the code you will see a line:

//#define DEBUG

around line 39 in the code. That is used to toggle "scaffolding" (debugging) code into and out of the program. For example, around line 250 you will see the statements:

```
#ifdef DEBUG
   Serial.print("i = ");
   Serial.println(minSWRs[i]);
#endif
```

If the *#define* on line 38 is uncommented (i.e., removing the "//" from the start of the line), the symbolic constant named DEBUG becomes defined, or active, in the program. The statements starting at line 250 can be verbalized as: "If the symbolic constant DEBUG is currently active, compile the statements from this line to the start of the *#endif* into the program. Now if you recompile the program, the two *Serial.print()* statements are compiled back into the program whereas they were not compiled into the program when the *#define DEBUG* statement was commented out. This provides an effective way to leave debug code in the program during testing, but not compiling those debug statements into the program for distribution...it "toggles" the debug code. If something goes south later on, just uncomment the *#define* and recompile the program and all the debug statements are once again in the compiled code.

The code provided here is only for your review. Use the *AAWithInterrupts.ino* file for compiling.

```
// Code written by Jack Purdum, W8TEE. It is for private use only and
// may not be distributed without written permission.
// Release 1.06
// Nov. 2, 2016

//The graphics package that I modified is taken from this package:
// Youtube video at:  https://www.youtube.com/watch?v=U5hOU-xxQgk

/*
  It requires and Arduino Mega (or UNO) and an Adafruit 3.5" TFT 320x480 + Touchscreen Breakout Board
  https://learn.adafruit.com/adafruit-3-5-color-320x480-tft-touchscreen-breakout/overview

  Adafruit libraries
  https://github.com/adafruit/Adafruit_HX8357_Library/archive/master.zip
  https://github.com/adafruit/Adafruit-GFX-Library/archive/master.zip

  optional touch screen libraries
  https://github.com/adafruit/Touch-Screen-Library/archive/master.zip
*/

// All the mcufriend.com UNO shields have the same pinout.
// i.e. control pins A0-A4.  Data D2-D9.  microSD D10-D13.
// Touchscreens are normally A1, A2, D7, D6 but the order varies
//
// This demo should work with most Adafruit TFT libraries
// If you are not using a shield,  use a full Adafruit constructor()
// e.g. Adafruit_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);

// Non-standard libraries may be found at the websites specified:

#include <AD9850SPI.h>        // https://github.com/F4GOJ/AD9850SPI
#include <EEPROM.h>           // Standard with IDE
#include <SD.h>               // Standard with IDE
#include <Adafruit_GFX.h>     // https://github.com/adafruit/Adafruit-GFX-Library
#include <MCUFRIEND_kbv.h>    // https://github.com/prenticedavid/MCUFRIEND_kbv
#include <Wire.h>
#include <Encoder_Polling.h>  //https://github.com/frodofski/Encoder_Polling
```

```c
//#define DEBUG          // Comment out if not debugging code

#define ELEMENTS(x) (sizeof(x) / sizeof(x[0]))
#define pulseHigh(pin) {digitalWrite(pin, HIGH); digitalWrite(pin, LOW); }

#define PORTRAIT  0                        // NOTE: These can set the origin to any corner of the
display with value 0 - 3
#define LANDSCAPE 1

#define MAXSAMPLES         100             // Max points on band sample
#define MAXPOINTS          75              // Max points read at each frequency

#define RIGHTMARGIN        70
#define LEFTMARGIN         40
#define BOTTOMMARGIN       60
#define TOPMARGIN          25
#define UPPERPLOTMARGIN    TOPMARGIN + 10
#define MINIMUMOFFSET      10

#define GRAPHAREATOPX      LEFTMARGIN
#define GRAPHAREATOPY      TOPMARGIN          // This is actually the menu width at top of display
#define GRAPHAREAORIGIN    h - BOTTOMMARGIN
#define GRAPHAREAORIGINY   GRAPHAREATOPX
#define GRAPHAXIXWIDTH     w - RIGHTMARGIN

#define MENUITEMWIDTH      100                // The pixel width of the background of main menu item
#define INTERMENUSPACING   25

#define MAXSCANPOINTS      100             // Number of plot data points

#define SWRMINSSET         0               // EEPROM minimum swrs set if this int = 1
#define SWRMINSADDRESS     2               // Starting EEPROM address for mins
#define NEXTSDFILENUMBER   90              // Two EEPROM bytes that holds the next number to be used
in a file name

#define EEPROMSTARTSCAN1   100             // Starting addresses for save scan data
#define EEPROMSTARTSCAN2   500

#define CW                 1
#define CCW                -1

#define PINA               18      // Encoder hookup; reverse A and B if it moves the wrong way
#define PINB               19
#define SWITCH             20

// following three pin definitions are needed by AD9850SPI library
#define FQ_UD              48      // connected to AD9850 freq update pin (FQ)
#define RESET              49      // connected to AD9850 reset pin (RST)
#define DATA               51      // connected to AD9850 serial data pin (MOSI)
#define W_CLK              52      // connected to AD9850 module word load clock pin (CLK)

#define ANALOGFORWARD      A6
#define ANALOGREFLECTED    A7

#define SD_SS              53      // SS pin used for SD card on LCD panel; connected through AA PCB
to Mega board

#define MAXFILES           20      // Number of SD files that can be opened
#define NAMELENGTH         13      // The max name size

#define FREQINCREMENT      100     // Used for upper/lower boundaries of scan adjustment
#define FIXEDFREQINCR      5       // Used in frequency adjustment measures

#define LOWER              1       // Used for lower band edge
#define UPPER              2       //          upper

char mySDFiles[MAXFILES][NAMELENGTH];

#ifndef min
#define min(a, b) (((a) < (b)) ? (a) : (b))
#endif


//=============================== Function Prototypes ===============
```

```c
void DrawBarChartHAxes(int flag);
void DrawBarChartH(int flag);
void DrawTable(int index);
char *Format(float val, int dec, int dig, char sbuf[]);
void FormatFrequency(float f, char buff[]);
float GenerateTestData(float x);
void GraphAxis(float gx, float gy, float w, float h, float xlo, float xhi, float xinc, float ylo, float
yhi, float yinc, char * title, char * xlabel, char * ylabel, unsigned int gcolor, unsigned int acolor,
unsigned int pcolor, unsigned int tcolor, unsigned int bcolor);
void GraphPoints(float x, float y, float gx, float gy, float w, float h, float xlo, float xhi, float
ylo, float yhi, unsigned int pcolor);
void PrintNextPoint(double currentFreq, int index);
int ReadSWRValue();
void runtests(void);
void ShowSubMenu(const char *menu[], int len);
//============================ Globals ============================
volatile int aVal;
volatile int encoderDirection;
volatile int dir;
volatile int pinALast;

// Colors

int BLUE    = 0x001F;
int GREEN   = 0x07E0;
int RED     = 0xF800;
int YELLOW  = 0xFFE0;
int WHITE   = 0xFFFF;
int BLACK   = 0x0000;
int DKGREEN = 0x03E0;
int LTPINK  = 0xFDDF;
int LTGREY  = ~0xE71C;
int MAGENTA = 0xF81F;


uint8_t latest_interrupted_pin;

int FwdOffSet;
int RevOffSet;

int encoderPassCount = 0;
int eepromMinIndex;
int filesFound;
int menuIndex;
int menuDepth;
int nextSDFileNumber;
int row;
int col;
int k;
int plotActive;
int w, h;
int spacing;
int scanMinX;
int scanMinY;
int switchState;
int swr[MAXSCANPOINTS];
int freq[MAXSCANPOINTS];
uint16_t g_identifier;
int16_t last, value;

// HF band edges follow...
int bandEdges[] = {1800, 2000, 3500, 4000, 5330, 5403, 7000, 7300, 10100, 10150, 14000, 14350, 18068,
18158, 21000, 21450, 24890, 24990, 28000, 29700};
int minSWRs[9];   // Each element is the minimum for the bands above
int pip[] = {60, 130, 200, 270, 340, 410};

float currentFreq, ox , oy;
float delta;
float hedge;
float bump;
float lastX;
float scanMinSWR;
float targetMinSWR[3];
```

```
// Menus
const char *menuLevel1[]  = {" Analysis ", " Options  ", " View Mins"};
const char *menuBands[]   = {"All", "160M", "80M", "60M", "40M", "30M", "20M", "17M", "15M", "12M",
"10M"};
const char *menuResults[] = {"Table"};
const char *menuLevel2[]  = {"New Scan", "Repeat", "Frequency"};
const char *menuFile[]    = {"Save Scan", "View Plot", "View Table", "Overlay", "Serial", "Delete
File"};


struct grafix {                                             // Graph structure declaration
  int x;                // upper left coordinate horizontal
  int y;                // upper left coordinate vertical
  int w;                // width of graph
  int h;                // height of graph
  float minX;           // minimum X graph value, can be negative
  float maxX;           // maximum X graph value
  float minY;           // minimum Y
  float maxY;           // maximum Y
  float xInc;           // scale division between lo and hi
  float yInc;           // y increment
  float currentValue;   // Current value
  int digitTotal;       // total digits displayed, not counting decimal point
  int decimals;         // digits after decimal point
  int barColor;         // Color for bar
  int voidColor;        // Background color in bar chart
  int backBar;          // Background bar color
  int border;           // Border color
  int textColor;        // Color for text
  int backFill;         // Background color for entire graph
  char label[30];       // Label text
} myG;

MCUFRIEND_kbv tft;                                          // Graph structure definition
File root;

/*****
  Purpose: To show a menu option

  Paramter list:
    const char *whichMenu[]     // Array of pointers to the menu option
    int len;                    // The number of menus

  Return value:
    void
*****/
void ShowMenu(const char *whichMenu[], int len) {
  int i;

  tft.setTextColor(WHITE, BLACK);
  for (i = 0; i < len; i++) {
    tft.setCursor(i * spacing, 0);
    tft.print(whichMenu[i]);
  }
  tft.setCursor(menuIndex * spacing, 0);
  tft.setTextColor(BLUE, WHITE);
  tft.print(whichMenu[menuIndex]);
  row = 0;
  col = menuIndex * spacing;
}

/*****
  Purpose: To read the minimum SWRs in  EEPROM memory

  Paramter list:
    void

  Return value:
    void
*****/

void ReadEEPROMMins()
{
  for (int i = 0; i < ELEMENTS(minSWRs); i++) {
```

```
      EEPROM.get(SWRMINSADDRESS + i * sizeof(int), minSWRs[i]);
#ifdef DEBUG
      Serial.print("i = ");
      Serial.println(minSWRs[i]);
#endif
  }
  EEPROM.get(NEXTSDFILENUMBER, nextSDFileNumber);   // The file number for the next SD file name
}

/*****
  Purpose: To prepare EEPROM memory for holding minimum scan values

  Paramter list:
    void

  Return value:
    void
*****/
void SetEEPROMMins()
{
  EEPROM.put(SWRMINSSET, 1);       // Says we've been here before
  for (int i = 0; i < ELEMENTS(minSWRs); i++) {
    EEPROM.put(SWRMINSADDRESS + i * sizeof(int), 0);
  }
}


/*****
  Purpose: This sets the default values for the graphics struccture

  Paramter list:
    void

  Return value:
    void
*****/
void SetGraphixDefaults()
{
  myG.x = 20;
  myG.y = 100;
  myG.w = 350;
  myG.h = 30;
  myG.minX = 1.0;
  myG.maxX = 3.0;
  myG.yInc = .25;
  myG.xInc = .25;
  myG.minY = 1.0;
  myG.maxY = 3.0;
  myG.digitTotal = 3;
  myG.decimals = 2;
  myG.barColor = GREEN;
  myG.backBar = DKGREEN;
  myG.border = GREEN;
  myG.textColor = WHITE;
  myG.backFill = BLACK;
}

/*****
  Purpose: Sign-on screen

  Paramter list:
    void

  Return value:
    void
*****/
void Splash()
{
  int row, col;
  tft.fillScreen(BLACK);

  row = h / 5;
  col = w / 4;
  tft.setTextSize(3);
```

```
    tft.setTextColor(MAGENTA, BLACK);
    tft.setCursor(6, row - INTERMENUSPACING * 2);
    tft.print(F("Milford Amateur Radio Club"));

    tft.setTextColor(RED, BLACK);
    tft.setCursor(col, row);
    tft.print(F("Antenna Analyzer"));
    tft.setTextSize(1);
    tft.setTextColor(WHITE, BLACK);
    col = w / 2;
    tft.setCursor(col, row + INTERMENUSPACING * 2);
    tft.print("by");
    col = w / 3;
    tft.setTextSize(2);
    tft.setTextColor(GREEN, BLACK);
    tft.setCursor(col - 20, row + INTERMENUSPACING * 4);
    tft.print(F("Jack Purdum, W8TEE"));
    tft.setCursor(col - 20, row + INTERMENUSPACING * 6);
    tft.print(F("Farrukh Zia, K2ZIA"));

    delay(3000);
}

/*****
  Purpose: To update a menu display.

  Paramter list:
    int whichWay    is the movement CW or CCW

  Return value:
    void
*****/
void AlterMenuOption(int whichWay)
{
  int oldColumn = col;
  int oldIndex = menuIndex;

  switch (whichWay) {
    case CW:
      menuIndex++;
      if (menuIndex == ELEMENTS(menuLevel1)) {
        menuIndex = 0;
        col = 0;
      }
      break;

    case CCW:
      menuIndex--;
      if (menuIndex < 0) {
        menuIndex = ELEMENTS(menuLevel1) - 1;
      }
      break;

    default:
      break;
  }
  col = spacing * menuIndex;
  tft.setTextColor(WHITE, BLACK);      // Erase old menu option
  tft.setCursor(oldColumn, 0);
  tft.print(menuLevel1[oldIndex]);
  tft.setCursor(col, 0);               // Show new menu option
  tft.setTextColor(BLUE, WHITE);
  tft.print(menuLevel1[menuIndex]);
}

#ifdef DEBUG
int freeRam() {
  extern int __heap_start, *__brkval;

  //Calculate the free RAM between the top of the heap and top of the stack
  //This new variable is on the top of the stack
  int l_total = 0;
  if (__brkval == 0)
```

```c
      l_total = (int) &l_total - (int) &__heap_start;
    else
      l_total = (int) &l_total - (int) __brkval;
    //
    l_total -= sizeof(l_total); //Because free RAM starts after this local variable
    return l_total;
}
#endif




/*****
  Purpose: To display and scroll data

  Paramter list:
    int swr[]   // The swr data
    int freq[]  // the frequency data

  Return value:
    void
*****/
void ShowAndScroll()
{
  int index = 0;
  encoderPassCount = 0;
  DrawTable(index);
  while (true) {
    if (digitalRead(SWITCH) == LOW)        // Return to main menu?
      break;

    encoderDirection = encoder_data(); // Check for rotation

    if (encoderDirection != 0)               // If it has rotated...
    {
      encoderPassCount++;                    // Need because there are 2 strobes per detent
      if (encoderPassCount == 2) {
        switch (encoderDirection) {
          case CW:
            index += 3;      // Showing 3 values per row
            break;
          case CCW:
            index -= 3;      // Showing 3 values per row
            break;
        }
        DrawTable(index);
        encoderPassCount = 0;
      }
    }
  } // end while (true)
}
/*****
  Purpose: To display the data from the most-recent scan

  Paramter list:
    int index           // Because list is scrollable, we need to track the index of the first display
element
    int swh[];         // Array of measured swr's. Note: Values must be divided by 100 to get swr.
Done to save memory
    int freq[];       // Array of associated frequencies

  Return value:
    void
*****/
void DrawTable(int index) {
  char buff[12];

  if (index < 0 || index > MAXSAMPLES - 42)   // There are 42 points shown on a screen
    return;
  tft.setTextColor(GREEN, BLACK);
  for (int k = 0; k < 14; k++) {
    tft.setCursor(0, k * 20 + TOPMARGIN);
    tft.print(((float) swr[index] * .01));
    tft.setCursor(70, k * 20 + TOPMARGIN);
    FormatFrequency(freq[index], buff);
```

```
    tft.print(buff);
    index++;
    tft.setCursor(160, k * 20 + TOPMARGIN);
    tft.print(((float) swr[index] * .01));
    tft.setCursor(230, k * 20 + TOPMARGIN);
    FormatFrequency(freq[index], buff);
    tft.print(buff);
    index++;
    tft.setCursor(320, k * 20 + TOPMARGIN);
    tft.print(((float) swr[index] * .01));
    tft.setCursor(390, k * 20 + TOPMARGIN);
    FormatFrequency(freq[index], buff);
    tft.print(buff);
    index++;
  }
}

/*****
  Purpose: To display the axes for a graph

  Paramter list:
    see list above...

  Return value:
    void
*****/
void GraphAxis(float gx, float gy, float w, float h, float xlo, float xhi, float xinc, float ylo, float
yhi, float yinc, char * title, char * xlabel, char * ylabel, unsigned int gcolor, unsigned int acolor,
unsigned int pcolor, unsigned int tcolor, unsigned int bcolor)
{
  char buff[10];
  int k;
  int f;
  float i;
  float temp;

  tft.fillScreen(BLACK);

  for (i = ylo; i <= yhi; i += yinc) {
    // compute the transform
    temp =  (i - ylo) * (gy - h - gy) / (yhi - ylo) + gy;

    if (i == 0) {
      tft.drawLine(gx, temp, gx + w, temp, acolor);
    }
    else {
      tft.drawLine(gx, temp, gx + w, temp, gcolor);
    }

    tft.setTextSize(1);
    tft.setTextColor(tcolor, bcolor);
    tft.setCursor(gx - 40, temp);
    // precision is default Arduino--this could really use some format control
    tft.println(i);
  }

  hedge = xlo;
  bump = (xhi - xlo) / 5.0;

  // draw x scale
  for (int i = 0; i < ELEMENTS(pip); i++) {
    temp = pip[i];
    tft.drawLine(temp, gy, temp, UPPERPLOTMARGIN, GREEN);
    tft.setTextSize(1);
    tft.setTextColor(tcolor, bcolor);
    tft.setCursor(temp - 10, gy + 10);
    f = (int) (hedge * .0001);
    FormatFrequency(f, buff);
    hedge += bump;
    tft.println(buff);
  }
  lastX = temp;
  //now draw the labels
  tft.setTextSize(1);
```

```cpp
    tft.setTextColor(acolor, bcolor);
    tft.setCursor(430, gy + 10);
    tft.println(xlabel);

    tft.setTextSize(1);
    tft.setTextColor(acolor, bcolor);
    tft.setCursor(5, gy - h - 15);
    tft.println(ylabel);

}

/*****
  Purpose: Break out frequency value into format: XX.XXX. This removes redundant "000" at end of
frequency

  Paramter list:
    float f          // The frequency to format
    char buff[]      // Where to store formatted result

  Return value:
    void
*****/
void FormatFrequency(int f, char buff[]) {
  char temp[11];
  int index, len;

  if (f < 1000) {                  // Under 40M
    itoa(f, buff, 10);
    temp[0] = buff[0];
    temp[1] = '.';
    strncpy(&temp[2], &buff[1], 3);
    temp[5] = '\0';
  } else {                         // Over 40M
    itoa(f, buff, 10);
    temp[0] = buff[0];
    temp[1] = buff[1];
    temp[2] = '.';
    strncpy(&temp[3], &buff[2], 3);
    temp[6] = '\0';
  }
  strcpy(buff, temp);
}

/*****
  Purpose: To format any floating point number. Little more than wrapper around dtostrf()

  Paramter list:
    float val              // THe number to format
    int dec                // The number of digits to display, including decimal point
    int dig                // Digits after decimal point
    char sbuf[]            // Where to put formatted result

  Return value:
    char *                 // Pointer to formatted result
*****/
char *Format(float val, int dec, int dig, char sbuf[] ) {
  int addpad = 0;
  char temp[dec + dig + 1];

  sbuf[0] = '\0';
  dtostrf(val, dec, dig, temp);
  int slen = strlen(temp);
  for (addpad = 1; addpad <= dec + dig - slen; addpad++) {
    strcat(sbuf, " ");
  }
  strcat(sbuf, temp);
  return sbuf;
}



/*****
  Purpose: To show a sub-menu that is below the main options
```

```
  Paramter list:
     const char *whichMenu[]      // Array of pointers to the menu option
     int len;                     // The number of menus

  Return value:
     void
*****/

void ShowSubMenu(const char *menu[], int len)
{
  int i;

  tft.fillRect(0, TOPMARGIN, w + 10, h, BLACK);     // Erase screen below top menu
  tft.setTextColor(GREEN, BLACK);
  for (i = i; i < len; i++) {
    tft.setCursor(col, i * INTERMENUSPACING + TOPMARGIN);
    tft.print(menu[i]);
  }
  tft.setCursor(col, TOPMARGIN);
  tft.setTextColor(BLUE, WHITE);
  tft.print(menu[0]);

  AlterMenuDepth(encoderDirection, menu, len);
  encoderDirection = aVal = CCW;
  encoderPassCount = 0;
  tft.fillRect(0, TOPMARGIN, w, h, BLACK);     // Erase screen below top menu
}


/*****
  Purpose: To highlight menu options as user scrolls through the list

  Paramter list:
     int whichWay         // Are we scrolling up or down
     const carh *menu[]   // The menu that is being scrolled
     int len              // The number of menu options

  Return value:
     void
*****/
void AlterMenuDepth(int whichWay, const char *menu[], int len)
{
  int oldRow = TOPMARGIN;
  int oldIndex;
  int itemCount = len;
  int ss;

  menuDepth = oldIndex = 0;

  while (true) {
    ss = digitalRead(SWITCH);
    if (ss == LOW) {
      return;
    }
    aVal = ReadEncoder();
    if (aVal != 0) {
#ifdef DEBUG
      Serial.println("In AlterMenuDepth 1");
#endif
      encoderPassCount++;                   // Need because there are 2 strobes per detent
      if (encoderPassCount == 2) {
        encoderPassCount = 0;
#ifdef DEBUG
        Serial.print("In AlterMenuDepth 2, passcount = 2");
        Serial.print("   aVal = ");
        Serial.println(aVal);
#endif
        oldIndex = menuDepth;
        switch (aVal) {
          case CW:
            menuDepth++;
            if (menuDepth == itemCount) {
              menuDepth = 0;
              row = TOPMARGIN;
```

```
          } else {
            row = menuDepth * INTERMENUSPACING + TOPMARGIN;        // Scroll to next menu item
          }
          break;

        case CCW:
          menuDepth--;
          if (menuDepth < 0) {
            menuDepth = itemCount - 1;
          }
          row = menuDepth * INTERMENUSPACING + TOPMARGIN;
          break;

        default:
          break;
      }
      tft.setTextColor(GREEN, BLACK);      // Erase old menu option
      tft.setCursor(col, oldRow);
      tft.print(menu[oldIndex]);
      tft.print("          ");
      tft.setCursor(col, row);                    // Show new menu option
      tft.setTextColor(BLUE, WHITE);
      tft.print(menu[menuDepth]);
      oldRow = row;
    }
   }
  }
}

/*****
  Purpose: To plot the X and Y axis for horizontal bar chart with appropriate tick marks and labels

  Parameter list:
    MCUFRIEND_kbv d        the graphics object
    struct grafix myG      the current state of that object
    int flag               if 1 labels are added, 0 if not

  Return value:
    void
*****/
void DrawBarChartHAxes(int flag)
{
  int offset = myG.x - 10;

  float stepval;
  float i, data;
  char buff[10];

  // draw the border, scale, and label once
  // avoid doing this on every update to minimize flicker
  // draw the border and scale

  tft.drawRect(myG.x , myG.y , myG.w, myG.h, myG.border);
  tft.setTextColor(myG.textColor, myG.backFill);

  // step val basically scales the hival and low val to the width
  stepval =  (myG.xInc * (float (myG.w) / (float (myG.maxX - myG.minX)))) - .00001;

  if (flag) {
    for (i = 0; i <= myG.w; i += stepval) {
      tft.drawFastVLine(i + myG.x , myG.y + myG.h + 1,  5, myG.textColor);
      // draw lables
      tft.setTextSize(1);
      tft.setTextColor(myG.textColor, myG.backFill);
      //      tft.setCursor(i + myG.x ,myG.y + myG.h + 10);
      tft.setCursor(i + offset , myG.y + myG.h + 10);

      // addling a small value to eliminate round off errors
      // this val may need to be adjusted
      data =  ( i * (myG.xInc / stepval)) + myG.minX + 0.00001;

      Format(data, myG.digitTotal, myG.decimals, buff);
      tft.println(buff);
    }
```

```
    }
}

/*****
  Purpose: To plot the X and Y values for the bar

  Parameter list:
    MCUFRIEND_kbv d          the graphics object
    struct grafix myG        the current state of that object
    int flag                 if 1 labels are added, 0 if not

  Return value:
    void
*****/

void DrawBarChartH(int flag)
{
  float level;
  char buff[10];

  // compute level of bar graph that is scaled to the width and the hi and low vals
  // this is needed to accompdate for +/- range capability
  // draw the bar graph
  // write a upper and lower bar to minimize flicker cause by blanking out bar and redraw on update

  if (myG.currentValue > 0.0 && myG.currentValue < 3.0) {
    level = (myG.w * (((myG.currentValue - myG.minX) / (myG.maxX - myG.minX))));
  } else {
    level = 0;
  }
  tft.fillRect(myG.x + level + 1, myG.y + 1, myG.w - level - 2, myG.h - 2,  myG.backBar);
  tft.fillRect(myG.x + 1, myG.y + 1 , level - 1,  myG.h - 2, myG.barColor);
  tft.setTextColor(myG.textColor, myG.backFill);                               // write the current
value

  tft.setTextSize(2);
  tft.setCursor(myG.x + myG.w + 10 , myG.y + 5);
  if (myG.currentValue > 0.0 && myG.currentValue < 3.0) {
    Format(myG.currentValue, myG.digitTotal, myG.decimals, buff);             // Changed to get rid
of String object
  } else {
    strcpy(buff, " N/A");
  }
  tft.println(buff);

  tft.setTextSize(2);
  switch (flag) {
    case 0:
    case 1:
      tft.setCursor(myG.x + 300, myG.y + 5);
      break;

    default:
      tft.setTextSize(2);
      tft.setCursor(myG.x , myG.y - 20);
      break;
  }
  tft.setTextColor(myG.textColor, myG.backFill);
  tft.println(myG.label);
}


/*****
  Purpose: To perform the scan options from the main menu
  Paramter list:
    void

  Return value:
    void
*****/
void ViewMinimums()
{
  int i, flag, len;
```

```
flag = 2;                       // Draw tick marks
len = ELEMENTS(menuBands);
//   col = MENUITEMWIDTH * 3;
col = spacing * 2;
ShowSubMenu(menuBands, len);

myG.x = 20;
myG.y = 100;
myG.w = 350;
myG.h = 30;
myG.minX = 1.0;
myG.maxX = 3.0;
myG.yInc = .25;
myG.xInc = .25;
myG.voidColor = BLACK;

myG.currentValue = ( (float) minSWRs[menuDepth - 1]) / 100.0;
strcpy(myG.label, menuBands[menuDepth]);
switch (menuDepth) {
  case 0:          // All
    myG.voidColor = GREEN;
    myG.y = 25;
    flag = 0;
    myG.h = 25;
    for (i = 1; i < ELEMENTS(menuBands); i++) {
      strcpy(myG.label, menuBands[i]);
      myG.currentValue = (float) minSWRs[i - 1] / 100.0;
      if (i == ELEMENTS(menuBands) - 1)
        flag = 1;
      DrawBarChartHAxes(flag);
      DrawBarChartH(flag);
      myG.y += 30;
    }
    break;
  case 1:            // 160
    DrawBarChartHAxes(flag);
    DrawBarChartH(flag);
    break;

  case 2:            // 80M
    DrawBarChartHAxes(flag);
    DrawBarChartH(flag);
    break;

  case 3:            // 40M
    DrawBarChartHAxes(flag);
    DrawBarChartH(flag);
    break;

  case 4:            // 30M
    DrawBarChartHAxes(flag);
    DrawBarChartH(flag);
    break;

  case 5:            // 20M
    DrawBarChartHAxes(flag);
    DrawBarChartH(flag);
    break;

  case 6:            // 17M
    DrawBarChartHAxes(flag);
    DrawBarChartH(flag);
    break;

  case 7:            // 15M
    DrawBarChartHAxes(flag);
    DrawBarChartH(flag);
    break;

  case 8:            // 12M
    DrawBarChartHAxes(flag);
    DrawBarChartH(flag);
    break;
```

```
      case 9:                    // 10M
        DrawBarChartHAxes(flag);
        DrawBarChartH(flag);
        break;

      default:
        break;
    }
}


/*****
  Purpose: To plot the points of a scan

  Paramter list:
    see list above...

  Return value:
    void
*****/
void GraphPoints(float x, float y, float gx, float gy, float w, float h, float xlo, float xhi, float
ylo, float yhi, unsigned int pcolor)
{
  byte flag = 0;
  int p = 0;

  if (y > oy) {
    flag = 1;
  }

  x =  (x - xlo) * ( w) / (xhi - xlo) + gx;
  y =  (y - ylo) * (gy - h - gy) / (yhi - ylo) + gy;

  if (y > UPPERPLOTMARGIN && x > gx) {
    tft.drawLine(ox, oy, x, y, pcolor);
    tft.drawLine(ox, oy + 1, x, y + 1, pcolor);
    tft.drawLine(ox, oy - 1, x, y - 1, pcolor);

    if (flag) {
      scanMinX = x;
      scanMinY = y;
    }
  }
  ox = x;                      // Save old coordinates so we know where line starts
  oy = y;
}

/*****
  Purpose: To set the lower and upper limits of a scan

  Paramter list:
    int whichOne        // Which edge to set: 1 = low, 2 = high

  Return value:
    int                // The ferquency of the edge
*****/

void SetBandEdge(int whichOne)
{
  int edge, offset;

  if (whichOne == LOWER) {
    tft.fillRect(0, TOPMARGIN, w + 10, h, BLACK);     // Erase screen below top menu
    tft.setCursor(0, TOPMARGIN);
    tft.setTextColor(WHITE, BLACK);
    tft.print(F("Set scan edges, defaults:"));
    tft.setCursor(0, TOPMARGIN + INTERMENUSPACING);
    tft.print(F("start: "));

    offset = TOPMARGIN + INTERMENUSPACING;         // Sets for lower frequency
    edge = bandEdges[menuDepth * 2];
  } else {
    tft.setCursor(0, TOPMARGIN + 2 * INTERMENUSPACING);
    tft.setTextColor(WHITE, BLACK);
```

```
      tft.print(F("  end: "));
      tft.setTextColor(GREEN, BLACK);
      tft.setCursor(80, TOPMARGIN + 2 * INTERMENUSPACING);
      tft.print( bandEdges[menuDepth * 2 + 1]);
      offset = TOPMARGIN + 2 * INTERMENUSPACING;    // Sets for upper frequency
      edge = bandEdges[menuDepth * 2 + 1];
  }
  tft.setTextColor(BLUE, WHITE);
  tft.setCursor(80, offset);
  tft.print(edge);
  encoderPassCount = 0;
#ifdef DEBUG1
    Serial.println("In SetBandEdge");
#endif

  while (digitalRead(SWITCH) == HIGH) {        // Wait for encoder switch change
    //     ReadEncoder();
    encoderDirection = encoder_data(); // Check for rotation
    if (encoderDirection != 0)            // If it has rotated...
    {
#ifdef DEBUG
    Serial.print("encoderDirection = ");
    Serial.print(encoderDirection);
    Serial.print("    edge = ");
    Serial.print(edge);
    Serial.print("  pass = ");
    Serial.println(encoderPassCount);
#endif
      encoderPassCount++;                     // Need because there are 2 strobes per detent
      if (encoderPassCount == 2) {
        switch (encoderDirection) {
          case CW:
            edge += FREQINCREMENT;      // Increased edge value
            break;

          case CCW:
            edge -= FREQINCREMENT;      // Decreased edge value
            break;
        }
        tft.setCursor(80, offset);
        tft.print(edge);
        encoderPassCount = 0;

        encoderDirection = 0;
      }
    }
  }

  if (whichOne == LOWER) {
    myG.minX = edge;
  } else {
    myG.maxX = edge;
  }
  tft.setCursor(80, offset);
  tft.setTextColor(GREEN, BLACK);
  tft.print(edge);
  delay(250);
}


/*****
  Purpose: To present the different options via menus

  Paramter list:
    void

  Return value:
    void
*****/
void NewScanOptions()
{
  int i, flag, len;
  int val;
  int saveIndex;
```

```
   float x, y;

   col = 0;
   if (plotActive == 0) {
     eepromMinIndex = menuDepth;
     len = ELEMENTS(menuBands) - 1;
     ShowSubMenu(&menuBands[1], len);
   }
   if (plotActive == 1)
     eepromMinIndex = menuDepth;

   if (plotActive == 0) {
     SetBandEdge(LOWER);                       // Set edges of scan
     SetBandEdge(UPPER);                       // Set edges of scan
   }

#ifdef DEBUG
   Serial.print("min = ");
   Serial.print(myG.minX );
   Serial.print("    max = ");
   Serial.println(myG.maxX);
#endif

   if (myG.minX < 1.0 || myG.minY < 1.0) {
     tft.setTextSize(2);
     tft.setTextColor(GREEN, BLACK);
     tft.setCursor(col - 20, row + INTERMENUSPACING * 4);
     tft.print("Band edges not set");
     return;
   }

   scanMinSWR = targetMinSWR[0] = 5.0;
   scanMinX = scanMinY = 0;

   myG.x = 60;                    //  CAUTION: Assumes minimum SWRs are in the vswrs[] array
   myG.y = 290;
   myG.w = 350;
   myG.h = 260;
   if (myG.maxX < 30000.0) {   // Need for repeat run with same parameters
     myG.minX *= 1000;
     myG.maxX *= 1000;
   }
   myG.xInc = 100000.0;
   myG.yInc = .25;

   ox = myG.minX ;
   oy = myG.maxY;

   GraphAxis( myG.x, myG.y, myG.w, myG.h,  myG.minX,  myG.maxX, myG.xInc, myG.minY, myG.maxY, .25,
"VSWR", "freq", "vswr", GREEN, RED, YELLOW, WHITE, BLACK);

   delta = (myG.maxX - myG.minX) / MAXSCANPOINTS;


   for (x = myG.minX, k = 0; x < myG.maxX; x += delta, k++) {
     PrintNextPoint(x, k);
     freq[k] = x * .001;
     y = (float) ( (float) swr[k]) * .01;
     if (y < 1.0 || x < 1.0)        // In case of garbage in array
       continue;
   }

   ox = myG.minX;
   oy = myG.maxY;
   scanMinSWR = 5.0;

   for (x = myG.minX, k = 0; x < myG.maxX; x += delta, k++) {
     y = (float) swr[k] * .01;
     x = (float) freq[k] * 1000;
     if (y < 1.0 || x < 1.0)        // In case of garbage in array
       break;
     GraphPoints(x, y, myG.x, myG.y, myG.w,  myG.h, myG.minX, myG.maxX, 1.0, 3.0, YELLOW);

     if (y < scanMinSWR) {
```

```
        scanMinSWR = y;
        targetMinSWR[0] = oy;
        targetMinSWR[1] = ox;
        lastX = x;
        saveIndex = k;
      }
   }
   lastX = ((myG.maxX - myG.minX) / (lastX - myG.x));
   lastX = lastX * (targetMinSWR[1] - myG.x) + myG.minX;
   val = scanMinSWR * 100.0;
   minSWRs[menuDepth] = val;                          // Update array
#ifdef DEBUG
   Serial.print("minIndex1 = ");
   Serial.print(eepromMinIndex);
   Serial.print("   plotActive = ");
   Serial.print(plotActive);
   Serial.print("   menuDepth = ");
   Serial.println(menuDepth);
#endif
   menuDepth *= sizeof(int);
   EEPROM.put(SWRMINSADDRESS + menuDepth, val);       // Update minimums in memory
   tft.setTextSize(2);                  // Plot header info
   tft.setTextColor(WHITE, BLACK);
   tft.setCursor(myG.x , myG.y - myG.h - 30);
   tft.print("Min SWR: ");
   tft.print(scanMinSWR);
   tft.print(" at Freq: ");
   tft.print((long) freq[saveIndex] * 1000);

   tft.setCursor(targetMinSWR[1] - MINIMUMOFFSET + 5, targetMinSWR[0] - 5);    // Min point
   tft.setTextColor(RED, BLACK);
   tft.print("+");

   plotActive = 1;

}


/*****
   Purpose: To perform the second menu option, "Options"

   Parameter list:
     void

   Return value:
     void
*****/
void DoOptions()
{
   int i, flag, len;
   int val;
   int saveIndex;
   int eepromAddr;
   int useFile;
   char fileName[13];
   char tempNum[5];

   float x, y;

   len = ELEMENTS(menuFile);
   col = spacing;
   ShowSubMenu(menuFile, len);
   strcpy(fileName, "SCAN");

   switch (menuDepth) {
     case 0:                                      // Save Scan
       tft.setTextColor(GREEN, BLACK);
       itoa(nextSDFileNumber, tempNum, 10);         // Build the new file name
       strcat(fileName, tempNum);
       strcat(fileName, ".CSV");
       tft.setCursor(50, 80);
       tft.print("Write new file: ");
       tft.print(fileName);
```

```
        root = SD.open(fileName, FILE_WRITE);
        if (!root) {
          tft.setTextColor(RED, BLACK);
          tft.setCursor(50, 120);
          tft.print("File open failure.");
          tft.setCursor(50, 145);
          tft.print("Is SD card inserted?");
          break;
        }
        WriteScanData(root);
        tft.setCursor(50, 145);
        tft.print("File named ");
        tft.print(fileName);
        tft.print(" successfully");

        nextSDFileNumber++;                                // Update for next new file
        EEPROM.put(NEXTSDFILENUMBER, nextSDFileNumber);    // Save in EEPROM
        break;

      case 1:                                             // View Plot
        int count;
        filesFound = ShowFiles();
        useFile = SelectFile();
        root = SD.open(mySDFiles[useFile], FILE_READ);
        tft.setTextColor(GREEN, BLACK);
        memset(swr, 0, sizeof(swr));
        memset(freq, 0, sizeof(freq));
        count = ReadScanDataFile(swr, freq, &myG.minX, &myG.maxX);

        myG.x = 60;                    //  CAUTION: Assumes minimum SWRs are in the vswrs[] array
        myG.y = 290;
        myG.w = 350;
        myG.h = 260;
        myG.minX *= 1000;
        myG.maxX *= 1000;
        myG.yInc = .25;

        ox = myG.minX;
        oy = myG.maxY;
        scanMinSWR = 5.0;

        GraphAxis( myG.x, myG.y, myG.w, myG.h,  myG.minX,  myG.maxX, myG.xInc, myG.minY, myG.maxY, .25,
"VSWR", "freq", "vswr", GREEN, RED, YELLOW, WHITE, BLACK);
        delta = (myG.maxX - myG.minX) / MAXSCANPOINTS;

        for (x = myG.minX, k = 0; x < myG.maxX; x += delta, k++) {
          y = (float) swr[k] * .01;
          x = (float) freq[k] * 1000;
          if (y < 1.0 || x < 1.0)         // In case of garbage in array
            break;
          GraphPoints(x, y, myG.x, myG.y, myG.w,  myG.h, myG.minX, myG.maxX, 1.0, 3.0, YELLOW);

          if (y < scanMinSWR) {
            scanMinSWR = y;
            targetMinSWR[0] = oy;
            targetMinSWR[1] = ox;
            lastX = x;
            saveIndex = k;
          }
        }
        lastX = ((myG.maxX - myG.minX) / (lastX - myG.x));
        lastX = lastX * (targetMinSWR[1] - myG.x) + myG.minX;
        val = scanMinSWR * 100.0;
        minSWRs[eepromMinIndex] = val;                     // Update array
        eepromMinIndex *= sizeof(int);

        tft.setTextSize(2);                                // Plot header info
        tft.setTextColor(WHITE, BLACK);
        tft.setCursor(myG.x , myG.y - myG.h - 30);
        tft.print("Min SWR: ");
        tft.print(scanMinSWR);
        tft.print(" at Freq: ");
        tft.print((long) freq[saveIndex] * 1000);
```

```
        tft.setCursor(targetMinSWR[1] - MINIMUMOFFSET + 5, targetMinSWR[0] - 5);    // Min point
        tft.setTextColor(RED, BLACK);
        tft.print("+");

        break;

    case 2:                                                // View Table
        filesFound = ShowFiles();
        useFile = SelectFile();
        root = SD.open(mySDFiles[useFile], FILE_READ);
        tft.setTextColor(GREEN, BLACK);
        count = ReadScanDataFile(swr, freq, &myG.minX, &myG.maxX);

        if (myG.minX < 1.0 || myG.minY < 1.0) {
          tft.setTextSize(2);
          tft.setTextColor(GREEN, BLACK);
          tft.setCursor(col - 20, row + INTERMENUSPACING * 4);
          tft.print("Band edges not set");
          break;
        }
        tft.fillScreen(BLACK);
        tft.setTextColor(GREEN, BLACK);

        scanMinSWR = targetMinSWR[0] = 5.0;
        scanMinX = scanMinY = 0;
        myG.x = 60;          //  CAUTION: Assumes minimum SWRs are in the vswrs[] array
        myG.y = 290;
        myG.h = 260;

        ox = myG.minX;
        oy = myG.maxY;
        delta = (myG.maxX - myG.minX) / MAXSAMPLES;
        int k;

        myG.xInc = 0;
        for (x = myG.minX, k = 0; x < myG.maxX; x += delta, k++) {
          y = (float) swr[k] * .01;
          x = (float) freq[k] * 1000;
          if (y < 1.0 || x < 1.0)        // In case of garbage in array
            break;
        }
        ShowAndScroll();
        break;


    case 3:                                        // Overlay
        //      int count;
        float overlayMinSWR;
        float overlayTargetMinSWR[2];
        float overlayLastX;
        int overlaySWR[MAXSCANPOINTS];
        int overlayFreq[MAXSCANPOINTS];

        filesFound = ShowFiles();
        useFile = SelectFile();
        root = SD.open(mySDFiles[useFile], FILE_READ);
        tft.setTextColor(GREEN, BLACK);
        count = ReadScanDataFile(overlaySWR, overlayFreq, &myG.minX, &myG.maxX);                      //
Read scan

        myG.x = 60;                      //  CAUTION: Assumes minimum SWRs are in the overlays[] array
        myG.y = 290;
        myG.w = 350;
        myG.h = 260;
        myG.minX *= 1000;
        myG.maxX *= 1000;
        myG.yInc = .25;

        ox = myG.minX ;
        oy = myG.maxY;

        scanMinSWR = 5.0;
        overlayMinSWR = 5.0;
```

```
      GraphAxis( myG.x, myG.y, myG.w, myG.h,  myG.minX,  myG.maxX, myG.xInc, myG.minY, myG.maxY, .25,
"VSWR", "freq", "vswr", GREEN, RED, YELLOW, WHITE, BLACK);
      delta = (myG.maxX - myG.minX) / MAXSCANPOINTS;
      overlayLastX = lastX;

      for (x = myG.minX, k = 0; x < myG.maxX; x += delta, k++) {
        y = (float) swr[k] * .01;
        x = (float) freq[k] * 1000;
        if (y < 1.0 || x < 1.0)         // In case of garbage in array
          break;
        GraphPoints(x, y, myG.x, myG.y, myG.w,  myG.h, myG.minX, myG.maxX, 1.0, 3.0, YELLOW);
        if (y < scanMinSWR) {
          scanMinSWR = y;
          targetMinSWR[0] = oy;
          targetMinSWR[1] = ox;
        }
      }

      tft.setCursor(targetMinSWR[1] - MINIMUMOFFSET + 5, targetMinSWR[0] - 5);    // Min point
      tft.setTextColor(RED, BLACK);
      tft.print("+");

      overlayMinSWR = 5.0;
      scanMinSWR = targetMinSWR[0] = 5.0;
      scanMinX = scanMinY = 0;
      ox = myG.minX;
      oy = myG.maxY;

      for (x = myG.minX, k = 0; x < myG.maxX; x += delta, k++) {
        y = (float) overlaySWR[k] * .01;
        x = (float) overlayFreq[k] * 1000;
        if (y < 1.0 || x < 1.0)         // In case of garbage in array
          break;
        GraphPoints(x, y, myG.x, myG.y, myG.w,  myG.h, myG.minX, myG.maxX, 1.0, 3.0, WHITE);

        if (y <= overlayMinSWR) {
          overlayMinSWR = y;
          overlayTargetMinSWR[0] = oy;
          overlayTargetMinSWR[1] = ox;
          lastX = x;
          saveIndex = k;
        }
      }
      lastX = ((myG.maxX - myG.minX) / (overlayLastX - myG.x));
      lastX = lastX * (overlayTargetMinSWR[1] - myG.x) + myG.minX;
      val = overlayMinSWR * 100.0;
      minSWRs[eepromMinIndex] = val;                        // Update array
      eepromMinIndex *= sizeof(int);

      tft.setTextSize(2);                                   // Plot header info
      tft.setTextColor(WHITE, BLACK);
      tft.setCursor(myG.x , myG.y - myG.h - 30);
      tft.print("Min SWR: ");
      tft.print(overlayMinSWR);
      tft.print(" at Freq: ");
      tft.print((long)lastX);

      tft.setCursor(targetMinSWR[1] - MINIMUMOFFSET + 5, targetMinSWR[0] - 5);    // Min point
      tft.setTextColor(RED, BLACK);
      tft.print("+");

      break;

    case 4:                                 // Serial monitor output
      filesFound = ShowFiles();
      useFile = SelectFile();
      root = SD.open(mySDFiles[useFile], FILE_READ);
      tft.setTextColor(GREEN, BLACK);
      count = ReadScanDataFile(swr, freq, &myG.minX, &myG.maxX);
      tft.fillScreen(BLACK);
      tft.setCursor(100, 100);
      tft.print("Writing to Serial port");

      for (k = 0; k < MAXSCANPOINTS; k++) {
```

```
        if (freq[k] < 1 || swr[k] < 1)        // In case of garbage in array
          break;
      }
      tft.setCursor(130, 200);
      tft.print("Done");

      break;

    case 5:                                        // Delete file

      filesFound = ShowFiles();
      useFile = SelectFile();
      if (ConfirmDelete(useFile)) {
        tft.fillScreen(BLACK);
        tft.setTextColor(GREEN, BLACK);
        tft.setCursor(100, 80);
        tft.print("Deleting file: ");
        tft.print(mySDFiles[useFile]);
        SD.remove(mySDFiles[useFile]);
        tft.setCursor(130, 125);
        tft.print("Deleted successfully");
        tft.setCursor(110, 150);
        tft.print("Press switch to continue:");
      }
      break;

    default:
      break;
  }
}

/*****
  Purpose: To confirm that the user really wants to delete this file

  Parameter list:
    int fileIndex        Index to name of the file selected for deletion

  Return value:
    int                  1 if they want to delete, 0 otherwise
*****/
int ConfirmDelete(int  fnToDelete)
{
  char optionDelete[] = " Delete ";
  char optionCancel[] = " Cancel ";
  int colOffset;
  int columnDelete = 100;
  int columnCancel = 200;

  tft.fillScreen(BLACK);
  tft.setTextColor(RED, BLACK);
  tft.setCursor(100, 80);
  tft.print("File to Delete: ");
  tft.print(mySDFiles[fnToDelete]);
  tft.setTextColor(GREEN, BLACK);
  tft.setCursor(columnDelete, 120);
  tft.print(optionDelete);
  tft.setTextColor(BLUE, WHITE);
  tft.setCursor(columnCancel, 120);
  tft.print(optionCancel);

  colOffset = columnCancel;
  encoderPassCount = 0;
  while (true) {
    aVal = ReadEncoder();              // Encoder is currently polling. Might replace with ISR
#ifdef DEBUG
    Serial.println("In ConfirmDelete");
#endif
    if (aVal != 0) {
      encoderPassCount++;                      // Need because there are 2 strobes per detent
      if (encoderPassCount == 2) {
        if (colOffset == columnCancel) {     // Switch to delete
          colOffset = columnDelete;
          tft.setTextColor(GREEN, BLACK);
          tft.setCursor(columnCancel, 120);
```

```
            tft.print(optionCancel);
            tft.setTextColor(BLUE, WHITE);
            tft.setCursor(columnDelete, 120);
            tft.print(optionDelete);
          } else {                                // Switch to cancel
            colOffset = columnCancel;
            tft.setTextColor(GREEN, BLACK);
            tft.setCursor(columnDelete, 120);
            tft.print(optionDelete);
            tft.setTextColor(BLUE, WHITE);
            tft.setCursor(columnCancel, 120);
            tft.print(optionCancel);
          }
          encoderDirection = aVal = CCW;
          encoderPassCount = 0;
        }
      }
    }
    switchState = digitalRead(SWITCH);
    if (switchState == LOW) {
      if (colOffset == columnDelete)
        return 1;
      else
        return 0;
    }
  }
}


/*****
  Purpose: To read the most-recent scan data to a CSV data file

  Parameter list:
    int *thisSWR       base of array for SWR
    int *thisFreq      base of array for frequency
    float *min          starting freq for scan
    float *max           ending freq for scan

  Return value:
    int            the number of data pairs read
*****/
int ReadScanDataFile(int *thisSWR, int *thisFreq, float * min, float * max)
{
  char temp;
  char buff[10];
  int i;
  int index = 0;

  i = 0;
  while (root.available()) {  // The first two values are min and max
    buff[i] = root.read();
    if (buff[i] != ',' && buff[i] != '\n') {
      i++;
    } else {
      if (buff[i] == ',') {      // Read SWR data
        buff[i] = '\0';
        *min = atof(buff);
        i = 0;
      } else {
        buff[i] = '\0';
        *max = atof(buff);
        i = 0;
        break;
      }
    }
  }

  i = 0;
  while (root.available() ) {
    buff[i] = root.read();
    if (buff[i] != ',' && buff[i] != '\n') {
      i++;
    } else {
      if (buff[i] == ',') {      // Read SWR data
        buff[i] = '\0';
        thisSWR[index] = atoi(buff);
```

```
          i = 0;
        } else {
          buff[i] = '\0';
          thisFreq[index++] = atoi(buff);
          i = 0;
        }
      }
    }
  }
  root.close();
  return index;
}

/*****
  Purpose: To save the most-recent scan data to a CSV data file

  Parameter list:
    File root      the currently-open file

  Return value:
    void

  CAUTION: This must be called AFTER a new scan is done
*****/
void WriteScanData(File root)
{
  char temp[10];
  int i;
  int xValMin, xValMax;

  xValMin = (int) (myG.minX / 1000.0);
  xValMax = (int) (myG.maxX / 1000.0);
  itoa(xValMin, temp, 10);          // Set frequency limits
  root.print(temp);

  root.print(",");
  itoa(xValMax, temp, 10);
  root.println(temp);

  for (i = 0; i < MAXSCANPOINTS; i++) {
    if (swr[i] == 0)
      continue;
    itoa(swr[i], temp, 10);
    root.print(temp);                    // Write data...
    root.print(",");                     // ...and a comma
    itoa(freq[i], temp, 10);
    root.println(temp);                  // Add data and newline
  }
  root.close();
}


/*****
  Purpose: To select a file from a list of the current SD files

  Parameter list:
    void

  Return value:
    int            an index into the file name array for the file to be used
*****/
int SelectFile()
{
  int dir, edge, offset;

  offset = TOPMARGIN + 25;
  edge = 0;

  for (int k = 0; k < filesFound; k++) {
    tft.setCursor(LEFTMARGIN + 80, offset + k * 25);
    tft.print(mySDFiles[k]);
  }

  tft.setTextColor(BLUE, WHITE);
  tft.setCursor(LEFTMARGIN + 80, offset);
```

```
      tft.print(mySDFiles[0]);
      encoderPassCount = 0;

    while (digitalRead(SWITCH) == HIGH) {   // Wait for encoder switch change
      //    ReadEncoder();
      dir = encoder_data(); // Check for rotation

      if (dir != 0)            // If it has rotated...
      {
        encoderDirection = dir;
        encoderPassCount++;                // Need because there are 2 strobes per detent
        if (encoderPassCount == 2) {
          tft.setTextColor(GREEN, BLACK);
          tft.setCursor(LEFTMARGIN + 80, offset + edge * 25);
          tft.print(mySDFiles[edge]);
          switch (encoderDirection) {
            case CW:
              edge++;        // Increased edge value
              break;

            case CCW:
              edge--;        // Decreased edge value
              break;

            default:
#ifdef DEBUG
              Serial.print("                           Souldn't be here,    edge = ");
              Serial.println(edge);
#endif
              break;
          }
          if (edge >= filesFound)
            edge = 0;
          if (edge < 0)
            edge = filesFound - 1;
          tft.setTextColor(BLUE, WHITE);
          tft.setCursor(LEFTMARGIN + 80, offset + edge * 25);
          tft.print(mySDFiles[edge]);
          encoderPassCount = 0;
          encoderDirection = 0;
        }
      }
    }
  }
  return edge;    // The index for the file name
}

/*****
  Purpose: To present a list of the current SD files

  Parameter list:
    void

  Return value:
    int          the number of files on the SD card
*****/
int ShowFiles() {
  int counter = 0;
  int offset;

  tft.setTextColor(GREEN, BLACK);

  root = SD.open("/");
  root.rewindDirectory();        // Begin at the start of the directory

  while (true) {
    File entry =  root.openNextFile();
    if (! entry) {
      break;
    }
    strcpy(mySDFiles[counter++],  entry.name());
    entry.close();
  }

  offset = TOPMARGIN + 25;
```

```
    tft.setCursor(0, TOPMARGIN);
    tft.print("SD files:");

    for (k = 0; k < counter; k++) {
      tft.setCursor(LEFTMARGIN + 80, offset + k * 25);
      tft.print(mySDFiles[k]);
    }
    tft.setCursor(0, offset + k * 25);
    tft.print("List Done!");
    root.close();
    return counter;
}


/*****
  Purpose: To read the current SWR for a given frequency

  Parameter list:
    float currentFreq     the frequency being tested
    int index             the scan point index into the array

  Return value:
    int           the number of files on the SD card
*****/
void PrintNextPoint(float currentFreq, int index) {
  float FWD = 0.0;
  float REV = 0.0;
  float VSWR;
  int   phase = 0; // phase for DDS.setfreq function in AD9850SPI library

  DDS.setfreq(currentFreq, phase);
  delay(10); // wait for AD9850 output to become stable

  swr[index] = ReadSWRValue();

#ifdef DEBUG
  Serial.print(currentFreq);
  Serial.print(" Hz VSWR: ");
  Serial.print(VSWR);              //Serial.print(int(VSWR*1000));
  Serial.print(", FWD: ");
  Serial.print(FWD);
  Serial.print(" REV: ");
  Serial.print(REV);
  Serial.print("   SWR: ");
  Serial.println(swr[index]);
#endif
}


/*****
  Purpose: To set up the default values for graphics structure
  Parameter list:
    void

  Return value:
    void
*****/
void InitGraphicsStructure()
{
  myG.x = 60;                  //  CAUTION: Assumes minimum SWRs are in the overlays[] array
  myG.y = 290;
  myG.w = 350;
  myG.h = 260;
  myG.minX = 7000;             // 40M is the default
  myG.maxX = 7400;

  myG.minX *= 1000;
  myG.maxX *= 1000;
  myG.yInc = .25;

  ox = myG.minX ;
  oy = myG.maxY;
}
```

```
/*****
  Purpose: To give a new scan choice to user

  Parameter list:
    void

  Return value:
    void
*****/
void DoNewScanChoice()
{
  int len;
  int localSWR;

  len = ELEMENTS(menuLevel2);
  col = 0;
  ShowSubMenu(menuLevel2, len);

#ifdef DEBUG
  Serial.println("In DoNewScanChoice()");
  Serial.print("At line 1973 cf = ");
  Serial.println(currentFreq);
#endif

  switch (menuDepth) {
    case 0:                                              // New scan
      plotActive = 0;      // Doing a new scan
      ox = myG.minX;
      oy = myG.maxY;
      NewScanOptions();
      delay(250);
      break;

    case 1:                                              // Repeat
      currentFreq = ox = myG.minX;
      oy = myG.maxY;
#ifdef DEBUG
      Serial.print("At line 1990 cf = ");
      Serial.println(currentFreq);
#endif
      NewScanOptions();
      delay(250);
      break;

    case 2:                                              // Frequency adjust
      len = ELEMENTS(menuBands) - 1;
      ShowSubMenu(&menuBands[1], len);
      SetFixedFrequencyBandEdge();
      delay(250);
      currentFreq = ox = myG.minX;
      oy = myG.maxY;
      currentFreq *= 1000.0;

#ifdef DEBUG
      Serial.print("At line 2005 cf = ");
      Serial.println(currentFreq);
#endif

      switchState = digitalRead(SWITCH);
      tft.fillRect(0, TOPMARGIN, w + 10, h, BLACK);    // Erase screen below top menu
      tft.setTextColor(GREEN, BLACK);
      tft.setCursor(50, 80);
      tft.print("Frequency: ");
      tft.setTextColor(WHITE, BLACK);
      tft.print(currentFreq);
      tft.setCursor(128, 120);
      tft.setTextColor(GREEN, BLACK);
      tft.print("SWR:");
      tft.setTextColor(WHITE, BLACK);

      DDS.setfreq(currentFreq, 0);
#ifdef DEBUG
      Serial.print("Bottom: currentFreq = ");
      Serial.println(currentFreq);
```

```
    #endif

        delay(10); // wait for AD9850 output to become stable


        while (switchState == HIGH) {               // Keep reading until a press
          DDS.setfreq(currentFreq, 0);
          delay(10); // wait for AD9850 output to become stable
          localSWR = ReadSWRValue();
          tft.setCursor(185, 120);
          tft.print(localSWR * .01);
          switchState = digitalRead(SWITCH);
          delay(250);
        }
        break;
    }
}

/*****
  Purpose: To set the normal band edges for a selected band

  Parameter list:
    void

  Return value:
    void
*****/
void SetFixedFrequencyBandEdge()
{
  int edge, offset;

  tft.fillRect(0, TOPMARGIN, w + 10, h, BLACK);     // Erase screen below top menu
  tft.setCursor(0, TOPMARGIN);
  tft.setTextColor(WHITE, BLACK);
  tft.print(F("Set scan edges, defaults:"));
  tft.setCursor(0, TOPMARGIN + INTERMENUSPACING);
  tft.print(F("start: "));

  offset = TOPMARGIN + INTERMENUSPACING;         // Sets for lower frequency
  edge = bandEdges[menuDepth * 2];

  tft.setTextColor(BLUE, WHITE);
  tft.setCursor(80, offset);
  tft.print(edge);
  encoderPassCount = 0;

  while (digitalRead(SWITCH) == HIGH) {          // Wait for encoder switch change
#ifdef DEBUG1
    Serial.println("In SetFixedFrequencyBandEdge");
#endif
    encoderDirection = encoder_data(); // Check for rotation
#ifdef DEBUG
    Serial.print("encoderDirection = ");
    Serial.print(encoderDirection);
    Serial.print("    edge = ");
    Serial.print(edge);
    Serial.print("  pass = ");
    Serial.println(encoderPassCount);
#endif

    if (encoderDirection != 0)               // If it has rotated...
    {
      encoderPassCount++;                      // Need because there are 2 strobes per detent
      if (encoderPassCount == 2) {
        switch (encoderDirection) {
          case CW:
            edge += FIXEDFREQINCR;        // Increased edge value
            break;

          case CCW:
            edge -= FIXEDFREQINCR;        // Decreased edge value
            break;
          default:
            break;
```

```
      }
      tft.setCursor(80, offset);
      tft.print(edge);
      encoderPassCount = 0;
      encoderDirection = 0;
    }
  }
}
  myG.minX = edge;
}

/*****
  Purpose: To read one bridge measurement

  Parameter list:
  void

  Return value:
  int          the swr * 1000 so it comes back as an int

  CAUTION: Assumes that frequency has already been set
*****/
int ReadSWRValue()
{
  int i;
  float sum[2] = {0.0, 0.0};

  float FWD = 0.0;
  float REV = 0.0;
  float VSWR;
  for (i = 0; i < MAXPOINTS; i++) {          // Take multiple samples at each frequency
    sum[0] += analogRead(ANALOGFORWARD);
    sum[1] += analogRead(ANALOGREFLECTED);
  }
  FWD = sum[0] / (float) MAXPOINTS;
  REV = sum[1] / (float) MAXPOINTS;

  if (REV >= FWD) {
    VSWR = 999.0;                            // To avoid a divide by zero or negative VSWR then set
to max 999
  } else {
    VSWR = ((FWD + REV) / (FWD - REV));      // Calculate VSWR
  }

#ifdef DEBUG1
  Serial.print("FWD = ");
  Serial.print(FWD);
  Serial.print("   REV = ");
  Serial.print(REV);
  Serial.print("   SWR = ");
  Serial.print(VSWR);
  Serial.print("   sum[0] = ");
  Serial.print(sum[0]);
  Serial.print("   sum[1] = ");
  Serial.println(sum[1]);
#endif

  return (int) (VSWR * 100.0);    // Save as an integer

}

/*****
  Purpose: To read the rotary encoder

  Parameter list:
    void

  Return value:
    int          the direction of the rotation,
                 0 = no rotation
                 1 = CW
                -1 = CCW
*****/
int ReadEncoder()
```

```
{
  int dir = encoder_data(); // Check for rotation

  if (dir != 0)              // If it has rotated...
  {
    encoderDirection = dir;
    aVal = dir;
    return aVal;
  }
}

//=================================================================================== setup()
========================================================================
void setup(void) {
  int eepromFlag;

#ifdef DEBUG
  Serial.begin(115200);
#endif

  pinMode(PINA, INPUT);
  pinALast = digitalRead(PINA);

  pinMode(PINB, INPUT);
  pinMode(SWITCH, INPUT_PULLUP);
  digitalWrite(SWITCH, HIGH);

  // attachInterrupt(digitalPinToInterrupt(PINA), ReadEncoder, RISING);
  encoder_begin(PINB, PINA); // Start the decoder


  // make sure internal pull up resistors on analog pins are disabled
  // so that they do not affect analog input readings
  pinMode(ANALOGFORWARD, INPUT);
  pinMode(ANALOGREFLECTED, INPUT);
  // enable MEGA 2.56V internal reference
  //analogReference(INTERNAL2V56);
  // After changing the analog reference,
  // the first few readings from analogRead() may not be accurate.
  // Read and discard 10 readings.
  for (int i = 0; i < 10; i++) {
    int fwd = analogRead(ANALOGFORWARD);
    int rev = analogRead(ANALOGREFLECTED);
  }

  DDS.begin(W_CLK, FQ_UD, RESET);
  DDS.calibrate(125000000);        // change this value if AD9850 calibration is needed
  DDS.setfreq(1.0, 0);             // set AD9850 output to 1 Hz and phase to 0
#ifdef DEBUG
  Serial.println ("DDS Module Initialized ...");
#endif

  menuIndex  = 0;                  // The highest level; drill down from there
  menuDepth  = 0;
  plotActive = 0;                  // We are not leading up to a plot

  g_identifier = tft.readID();     // Get TFT ID  3.95" = 0x9486, 3.6" = 0x9488

  if (g_identifier == 0x9486) {    // If 3.6" TFT, invert colors
    BLUE    = ~BLUE;
    GREEN   = ~GREEN;
    RED     = ~RED;
    YELLOW  = ~YELLOW;
    WHITE   = ~WHITE;
    BLACK   = ~BLACK;
    DKGREEN = ~DKGREEN;
  }

  tft.begin(g_identifier);
#ifdef DEBUG
  Serial.print("TFT ID ");
  Serial.println(g_identifier, HEX);
#endif
```

```
    EEPROM.get(SWRMINSSET, eepromFlag);      // Have the minimum SWRs been set?

    if (eepromFlag != 1) {
      SetEEPROMMins();
    } else {
      ReadEEPROMMins();
    }

    tft.setRotation(LANDSCAPE); // In Landscape mode, the arguments are: setCursor(col, row) with botton
row = w - 20.
    w = tft.width(),
    h = tft.height();
    InitGraphicsStructure();

    spacing = (w - 20) / ELEMENTS(menuLevel1);       // Spacing between main menu items
    tft.setTextSize(2);
    SetGraphixDefaults();

    Splash();
    tft.fillScreen(BLACK);
    ShowMenu(menuLevel1, ELEMENTS(menuLevel1));

    tft.setTextColor(GREEN, BLACK);
    tft.setCursor(50, 120);
    if (!SD.begin(SD_SS)) {
      tft.println("initialization failed!");
      tft.setCursor(50, 145);
      tft.println("Is SD card in slot?");
    }
}

//================================================================================ loop()
===============================================================================

void loop(void) {
  int i, flag, len;
  int val;

  float x, y;

  aVal = ReadEncoder();                  // Encoder is currently polling. Might replace with ISR

  if (aVal != 0) {
    encoderPassCount++;                        // Need because there are 2 strobes per detent
#ifdef DEBUG
    Serial.print("encoderPassCount = ");
    Serial.print(encoderPassCount);
    Serial.print("   aVal = ");
    Serial.print(aVal);
    Serial.print("   encoderDirection = ");
    Serial.println(encoderDirection);
#endif
    if (encoderPassCount == 2) {
      AlterMenuOption(encoderDirection);
      encoderDirection = aVal = 0;
      encoderPassCount = 0;
    }
  }

  switchState = digitalRead(SWITCH);
  if (switchState == LOW) {
    delay(250);           // No debounce on switch
    menuDepth++;          // Add depth check later
    row += 50;            // Do row check later
    switch (menuIndex) {

      case 0:           //====================================== New Scan
        DoNewScanChoice();
        break;

      case 1:           //====================================== Options
        DoOptions();
        break;
```

```
      case 2:                //========================================  Min Options
        ViewMinimums();
        break;

      default:
#ifdef DEBUG
        Serial.print(F("I shouldn't be here: menuIndex = "));
        Serial.println(menuIndex);
#endif
        break;
    }

    while (digitalRead(SWITCH) == HIGH) {     // Force a pause to view results of above menu
selection...
      switchState = HIGH;
    }
    tft.fillScreen(BLACK);
    menuDepth = 0;
    ShowMenu(menuLevel1, ELEMENTS(menuLevel1) );
  }
}
```

Appendix B: List of Major Parts for AA

Resistors

Ohms	Color Code	Schematic Part Number
10	Brown Black Black	R19
51	Green Brown Black	R1, R2, R3, RL
1K	Brown Black Red	R6, R10, R13, R15, R17
2K	Red Black Red	R12, R14, R16, R18,
10K	Brown Black Orange	R4, R7, R8, R11
100K	Brown Black Yellow	R5, R9

Capacitors

NF	Standard Number	Schematic Part Number
10	103	C4, C5
100	104	C2, C3, C3, C6, C7, C8

Other Parts

Description	Details
Arduino Mega 2560 Pro Mini	Those sold by Banggood are cheaper, but don't fit the PCB. We buy from vendor wideenm991 on eBay.
AD9850 module	Type II. Do *NOT* buy a Type 1. See Figure 22.
Rotary encoder	KY-040A. Check for threaded shaft
DDS-VFO PCB	QRPGuys.com
VR-MINI_360 Buck converter	1
1N4001 diodes	4
3.5" TFT display, produced by mcufriend.com	Ebay #172522909064 or similar.
SPDT toggle switch	Standard miniature center off
100uH axial inductor	http://www.taydaelectronics.com/
MAR-3SM	MSA-0386 is getting harder to find
Project case	Banggood has many to choose from
Power connector	Choice depends on your case selection. Some use a battery pack inside the case.
LM358	Op amp
AA143 germanium diode	Ebay

6 pin header socket	2
8 pin header socket	5
40 pin header strips	10. You don't need this many, but you'll use them
Dupont jumper wires	(F-F) 8" length (Quantity 20)
Misc	Bolts, nuts, wire, solder, etc.