Using an Arduino

to Automatically Tune

an MFJ-1788 Magnetic Loop Antenna

and Elecraft KX3 Transceiver

By Elwood Downey, WBØOEW

15 Sept 2015

Abstract

This article describes a microprocessor controlled system that continuously monitors the transmit frequency of an Elecraft KX3 transceiver (or similar) and automatically keeps an MFJ-1788 magnetic loop antenna in proper tune without any operator interaction with the antenna. The operator is free to operate the KX3 in the normal manner, but if the system notes the frequency has changed beyond the bandwidth of the antenna, the system temporarily reroutes the RF path in order to measure return loss while rotating the antenna capacitor for a proper match, then returns the RF path back to the radio to resume normal operation. No operator action is required while this tuning is under way. No modifications whatsoever are required in either the KX3 or the MFJ-1788.

I enjoy using my MFJ 1788 mag loop antenna[i] and KX3 transceiver[ii] together. I appreciate the effort MFJ put into their loop controller and think it is a clever and effective design. However, because the loop is very narrow band, I find it awkward and distracting to frequently retune after even a small QSY. I was aware that other antennas, such as the SteppIR, connect to the KX3 to monitor the operating frequency and retune automatically as necessary and I wanted to have the same convenience with my loop.

Another motivation was a way to tune without transmitting. I was wary of using even low power during the tuning process which can take many seconds for concerns about the stress this might be causing my rig. Plus, since the tuning process is performed so often, it creates QRM which I prefer to avoid. And if I could tune without transmitting, it would work over the full range of the antenna, not just in the legal ham bands.

This paper describes a solution that meets both challenges. I first discuss the design process I went through, then the build process, then describe how to operate the device and close with some observations. I hope that there is enough detail included so you can understand the design principles involved but also so you can adapt the design to your own situation.

Disclaimer: While I am happy to share what I have done and will be glad to discuss the project with anyone, if you try this yourself and break anything, including your antenna, your rig or yourself, **you do so entirely at your own risk and I am not responsible**. The design presented here is specifically for the MFJ-1788 and the Elecraft KX3 and has only been tested on my particular units. It may well be that it can be adapted to other similar equipment or even other uses entirely and you are welcome to do so, but all such uses are the sole responsibility of the user. Ok, back to the fun.

1. The Design

We begin by going over the process I went through to design the new antenna controller.

1. Requirements

Any design should begin with a list of requirements. My list of detailed requirements boil down to the following:

1. Be able to monitor the KX3 ACC1 serial communication to read the radio transmit frequency in a manner that is transparent to, and thus does not interfere with, its normal usage as a connection to a computer or KXPA100 amplifier. This implies that I wanted the connection to be entirely passive without the need for any polling from the new controller.
2. Be able to tune the antenna without transmitting with the KX3.
3. Use PWM (Pulse Width Modulation) to control speed and direction of the DC motor that turns the antenna loop capacitor.
4. Be able to sense the antenna motor end-of-travel in case no peak is found in a given direction.
5. Provide several switches and potentiometers for operator inputs.
6. Provide several status LEDs to keep the operator informed of state information.
7. Include sufficient computing power to monitor the radio frequency on a timely basis, decide when it is necessary to retune, and control the motor to find a peak within several seconds from start of search.

8. Include provision for a coax relay to automatically switch the antenna to the tuning controller while searching, and back to the radio when completed without operator intervention.

9. Make no changes whatsoever to either the radio or the antenna so they can be returned to service at any time in their original condition.

Before arriving at these requirements, I explored other approaches. I settled on searching for maximum return loss but I also tried searching for maximum receive noise. This uses much less hardware: no RF generator, bridge or detector; just send the audio to an ADC, but I was never satisfied. It worked very well when in the clear but the main challenge was making it work in the presence of a signal. My best approach was to use an FFT and use only those frequencies with minimal strength (to measure only the noise and avoid the wildly varying modulation content). I got pretty close but, again, was never satisfied. I still think it would be cool if it worked though. I also tried using the SWR meter built into the KX3. The main problem with that is it's not very repeatable so there were many false nulls.

2. Choice of Microprocessor

After researching the available options I settled on an Arduino Uno for the main processor. It is inexpensive, has a rich ecosystem of development tools and supporting information and is rapidly gaining traction as a preferred platform in amateur radio circles. It comes well equipped to address all the requirements listed above with some additional hardware.

I wrote the software, called “sketches” in Arduino parlance, in small steps to understand each requirement separately. The tools and techniques for doing so are covered well elsewhere[iii] and will not be repeated here.

3. Monitoring the KX3 Frequency

The new tuner needs to constantly monitor the KX3 frequency coming from its ACC1 connection. I dug into both the KX3 and Uno schematics to see how I could listen to this line without interfering with its normal use with a computer. Once I realized that the polarity between the two units was opposite (the KX3 uses RS232 with mark High whereas the Arduino uses TTL with mark Low) it was a simple matter to wire up a transistor to serve as both an isolation buffer and level inverter (Q6 in the schematic).

In order to share the serial connection for use with the Arduino IDE Serial Monitor at the same time as it listens to the KX3, you must set the IDE to 38400 baud to match the radio. Note that this speed for the Arduino Serial Monitor only became available starting with IDE Version 1.6.1[iv].

After more study of the Arduino (really the Atmel) serial port function and the KX3 Programmers Manual for the command syntax, I had code running that reliably recognized and extracted the operating frequency as I turned the main knob or changed bands on the KX3.

Note I decided not to use the Arduino Serial class because I wanted a fully interrupt-driver serial mechanism that could effectively run simultaneously “in the background” while my main loop

was controlling everything and also because I did not want to poll the radio and worry that any latency in my main loop would risk missing characters. The results are a little more complex but functionally is the same and works fine with the IDE Serial Monitor.

Finally on this topic, with factory default settings the KX3 only reports its frequency to the ACC1 port when polled but this was counter to my requirement of a non-invasive read-only connection. Turns out in the KX3 menu system is an option called AUTO INF which can be set to ANT CTRL to create exactly the desired behavior — clearly the folks at Elecraft anticipated this use case. With this setting, any time the operator changes frequency or switches bands the KX3 automatically sends a new frequency report within a second or so even when not connected to a computer. However, if you have your KX3 connected to computer software that gets confused with these unsolicited responses, it is pretty safe to assume said software is surely doing its own polling so turn off this menu setting, let the software do its own polling and everything should still work fine.

4. Controlling the Antenna Motor Speed

The next requirement I tackled was controlling the loop motor. I was familiar with the idea of Pulse Width Modulation whereby the effective power of a digital signal is controlled by changing the fraction of time during which it remains at a logic High level while maintaining a constant frequency, otherwise known as changing its duty cycle. So that was a simple means to control the speed but I also needed a way to change direction. After more study, I found this is normally done by an H Bridge. This is a classic circuit that connects the two wires from a DC motor

with four SPST switches, arranged in such as way that the proper combination of switch states can connect either side of the motor to power or ground potential, thus providing a means to change the polarity of the motor and thus its direction of rotation. This technique is so common, in fact, that dedicated ICs are available that perform this function with just a few parts. I did not have such an IC, but I had enough parts in my junk box to fabricate an H Bridge from first principles. I built up a circuit and my next trial sketch convinced me I could control both motor speed and direction using two PWM outputs from the Arduino.

5. Detecting End-of-Travel

The next requirement was to detect the end of travel. In the MFJ-1788, although in principle the capacitor mechanism could rotate endlessly without doing any harm, in fact it is only allowed to rotate one half revolution. I can imagine this design decision was made because the other half rotation does not really provide any new values of capacitance, and because the net capacitance would first increase and then decrease while the motor continues to turn in one direction which would be quite confusing to any tuning algorithm. The rotation limits are implemented as two physical switches and diodes inside the antenna module. A given switch opens when motion in a given direction reaches its end of travel and yet the diode allows current to flow in the opposite direction even with the switch open.

The MFJ-1788 makes use of a Bias-T to provide motor power through the same coax as the RF. This is a clever way to eliminate an extra control cable, but it also means the state of these limit switches is not directly available to the control end of the system. Fortunately, the only function

these switches effectively perform is interrupt the path of current to the motor. Thus the activation of a limit switch can be detected by simply monitoring the current to the motor and noting if it drops to zero. I did this by using a pair of optoisolators, a pair being required in order to allow for both polarities sent to the motor for control of direction. Although it is true this technique does not directly provide the means to know *which* limit switch was activated, this can be reliably inferred by knowing which way the motor is being commanded to move when the current stops.

However, this method of detecting a travel limit comes with one caveat. Using PWM means that the current is intentionally brought down to zero during each cycle. So a means was needed for this frequent occurrence of zero current to not cause a false indication of a limit. The solution turned out to be as simple as adding a timer to the limit detection algorithm. Although the instantaneous current detector circuit still reports no current during each PWM cycle, a *logical* limit is not reported unless the condition persists for some small length of time.

6. Operator Inputs

In this application, the requirement to provide operator command inputs is sufficiently simple as to be met with some small momentary contact push button switches. I ended up using three:

- Tap to force an automatic search to commence, if desired
- Hold to manually rotate lower in frequency, release to stop
- Hold to manually rotate higher in frequency, release to stop

There are other combinations of switch inputs for lesser used functions which will be described later. Each switch is connected directly between a digital input and ground, and the Arduino is programmed to provide an internal pullup resistor to the positive supply rail, a handy feature that saves a resistor for each switch. Note in the code the logic is inverted, such that a High denotes the switch is idle and a Low denotes the switch is being pressed.

In addition, there are two analog values that must be set based upon your particular antenna characteristics. Even with my single unit, I find it necessary to adjust these whenever the weather temperature or humidity changes appreciably, probably because of their effect on the motor lubrication and friction in the simple journal bearings. The operator adjusts these by turning two potentiometers. One is called Slew, which sets the fast slewing rate. The other is called Step, which sets the fine stepping pulse duration time. These will be explained more fully later when we discuss the tuning algorithm. These pots also have specialized applications as explained later.

7. LED State Indicators

I decided that controller state information can be communicated clearly enough using four colored LEDs. At times during development I found it handy to connect an 8 character 8x8 LED matrix array controlled using the SCI bus, but this is not needed for the final system. Each LED is connected to a digital output pin with a series resistor to ground. I chose to define the four LEDs as follows:

- light a red LED while a logical end-of-travel limit is being detected

- light a yellow LED to indicate the antenna is moving down in frequency
- light a blue LED to indicate the antenna is moving up in frequency
- blink a green LED when communication with the KX3 has reliably determined the operating frequency.

As with the switches, other LED combinations are used for lesser used functions as described later.

8. Measuring Antenna Match

The final piece of the puzzle is a way to measure the degree of impedance match of the antenna. I considered many approaches but finally settled on using a balanced bridge circuit to sense when the magnitude of the input impedance of the antenna connector was near 50 ohms. The idea is to connect three 50 ohm resistors into a square, connect the antenna in the place of the fourth resistor, then drive two opposite corners with a reference frequency and measure the voltage across the two remaining corners. The lower the measured signal across the bridge, the better the antenna impedance agrees with the surrounding resistor values. Such a bridge does not provide the sign of the reactance when not matched but we can infer that in other ways.

With this design in mind, it remains to choose a signal source and a detector. It turns out there are two ICs from Analog Devices that serve both these goals admirably. The AD8307 log amp[v] is an excellent detector and the AD9851 Direct Digital Synthesize[vi] is a flexible RF signal source.

These parts have been RF work horses for some time so the only real decision was how best to proceed with an implementation around them.

A little searching on the web finds lots of schematics and tips for using these parts in similar applications. Being an experimenter I was tempted to start with bare parts and work up my own solution. But then I found a commercial implementation that was just too good to ignore. The FoxDelta AAZ-0914A Antenna Analyzer[vii] provides a complete kit for exactly this purpose. For about $50, there was no way I could match this with my own parts and time. All you do is connect the antenna, 5 VDC, a few wires to control the chips and the Arduino has an easy time of measuring the degree of mismatch anywhere in the frequency range of the antenna.

Note that the FoxDelta unit includes a PIC processor to provide a nice USB interface to a host computer. However, rather than work through an intermediate control layer, I decided to unplug the PIC and attach leads directly from each control chip to some Arduino IO pins. This allowed me to control the Analog Devices chips exactly as I wanted, and also makes the software independent of the AAZ-0914A and thus entirely suitable for others who might want to use a different implementation using these chips.

9. Tuning Algorithm

So now that we can read the radio frequency, control the motor, interact with the operator and measure the degree of tuning match it's time to consider the heart of the matter: the automatic tuning algorithm. This involved a large amount of experimenting and carefully observing how

the motor reacts to commands. I'll spare you all the false starts but one lesson stands out of paramount importance: there is a large amount of backlash in the drive train and it is not particularly consistent with direction, speed or position. The MFJ design includes a spring that probably reduces this to some extent, but the very low power motor precludes it from providing much in the way of compensating force and thus its effectiveness is marginal at best.

Thus, dealing with the backlash becomes the primary challenge. The tuning technique I finally settled on is the following:

1. Keep comparing the radio frequency with the last known antenna tuning frequency. Using an estimate of the antenna Q (which can be easily adjusted in the source code if desired), determine when the two values different by an amount that is worth correcting. The code takes into account that the useful bandwidth of the loop antenna is smaller at lower frequencies. When the two values are sufficiently far apart, decide which way the motor needs to rotate and go to step 2.
2. Begin by slewing full speed in the opposite direction for a short while. Yes, you heard that right. By going a goodly ways in the wrong direction first, we are assured of always passing through the best match position at full speed, even if it is very close to the starting position. I tried many techniques to approach more slowly but this turned out to be by far the most reliable. After this brief burst, stop, slew at full speed in the other (correct) direction and go to step 3. It is the Slew pot that defines this fast rate of motor rotation.

3. While going full speed toward the desired match position, measure the match as quickly as possible. Given that the match can be read very rapidly, we are assured of a steep and deep dip as we pass through the best match value. As we are performing this search, record the best match value seen so far to be used later. When the dip is detected, stop the motor. Another benefit of always slewing rapidly through the best match position is that we can be assured the motor has definitely overshot the best position. If we had approached more slowly, we may or may not have passed the best position and so we really would not have learned very much about our target. Now that we know for sure we are stopped just passed the best position, go to step 4.

4. Now the fun begins. We make very small steps and measure the match after making sure the motor has come to a complete stop after each step. The duration that power is applied for each step is controlled by the Step pot. It took me days to realize that when power is removed from the motor the capacitor can continue to change aimlessly, even though I inspected the coupling bushing in the antenna to insure it is tight. Measuring the antenna match during these uncertain motions is at best meaningless and at worst just adds to the confusion. So during this procedure, after power it removed for each step, the match is measured repeatedly until it no longer changes. Only then do we believe the motor and capacitor have really stopped and the measurement is meaningful. By insuring a full stop, each of these final measurements are reliable and we can repeat making small steps looking for the next dip. The next step describes exactly what we are looking for.

5. Unlike the dip seen while performing the initial fast slew, in this phase we must minimize the degree of overshoot through the best match condition. This is because since we cannot actually measure position, there is no way to go back to a known position in a reliable fashion. Thus we need a way to detect we are just barely passed the best match position. To do this, as we make each (stopped) measurement, we look for a match reading that is better than the best reading found during the slew in step 3 followed by a larger value. This strongly suggests we are exactly one step passed the best match position. We could stop here and be pretty close but we go on to step 6.

6. To review the work up to this point, we slewed rapidly enough through best match to insure overshoot and we recorded the best value seen during that run. We have turned around and found that same match again but going much more slowly so we know we are close. What we do now is basically repeat step 5 but at half the step time. We keep doing this until we are all but completely stopped. So in summary, we walk back and forth over the best match going more and more slowly, the plan being that when we go so slow as to be stopped, we should be almost exactly at the best match position.

All the while this is going on, we also watch for end-of-travel, in which case we turn around and start over. We also watch for the operator to tap any of the switches in which case we abort the tuning attempt and stop where we are.

2. The Build

Now that I had a design and major implementation details worked out, it was time to build it. By the time I had everything working my bench was pretty much covered with several breadboards and a large number of wires running everywhere but I love this stuff so that was a feature of the project, not a bug!

1. Construction

For the following, it is helpful to refer to the schematic diagram[viii].

Many of my component choices and implementation decisions were based simply on using suitable parts I had on hand as much as possible, even if perhaps there are simpler choices. Looking at the final schematic, you might guess correctly that I had a large number of 2N3904 and 2N3906 transistors, 1N4148 diodes and MCT6 dual optocouplers, so naturally that's what I used. These transistors are rated for only about 200 mA collector current but, remarkably, the MFJ motor draws only about 10 (yes 10) mA so even these small signal parts work fine in what otherwise would require larger current devices. Even the LEDs in the optocoupler used as a current sense can handle 60 mA of continuous current. I measured about 40 volts of back EMF so the 60 V maximum reverse voltage of the 1N4148 used here as clamping diodes is also adequate. But, again, feel free to use what you prefer.

The most obvious effect to use stuff on hard is that I made my own H Bridge from discrete components. If you have, or want to purchase, an H Bridge chip such as the TI L293D or even a

complete Arduino motor control shield such as those available from Adafruit or Sparkfun, by all means do so. The big point here, however, is to take note that *the Bias-T requires that the power supply for the antenna motor most have both sides isolated from station ground*. The reason is that either side of this isolated supply can, depending on the desired motor direction, be connected to station ground. I could have worked up an isolated supply, or just used a commodity wall-wart, but given the remarkably low current draw of the motor, I just chose to power the H Bridge with batteries adding up to 12 volts. I am still using the original set I started with but note that if your unit stops operating reliably, measuring the battery voltage should be a first thing to check. Generous use of the optocouplers made easy work of connecting the grounded Arduino to the floating H Bridge.

Both the Arduino and the Fox Delta require a clean supply of 5 V DC. For this I found an inexpensive variable buck converter on Amazon that draws from the 13.8 V main supply for the unit.

Two PWM outputs from the Arduino drive the H Bridge. They turn on either Q1 and Q4, thus connecting the left and right leads of the motor to positive and negative supply, respectively, or Q2 and Q3 which provides the opposite polarity to the motor. The motor is off when all four drivers are open. Note the supply would be shorted if both left or both right drivers were on at the same time, but the logic of the optocoupler wiring, U1 and U2, makes this impossible, thankfully making the design immune at least from this programming error.

Rather than build my own Bias-T, I just purchased the MFJ-4116[ix]. I figured since they use the same design within their antenna, their separate product would probably be compatible and that

proved to be the case. Doing so certainly saved some effort and perhaps a little money. My only regret is the poor quality SO-239 connectors they used in my unit. But I digress.

The motion limit sensor uses another pair of optocouplers in U3 wired to connect one Arduino digital input to ground while current flows in either direction, causing the Arduino to read a Low. If current does not flow the coupler outputs will both be open and the Arduino will read a High logic value. As described above, a little extra care is required in the software because the current is also zero in between PWM pulses.

Several IO lines connect to the Fox Delta unit. I did this by carefully removing the original PIC (and packing it for safe keeping if I ever want to use it again) and replacing it with two header strips. I then plugged male bread board jumper wires into the female header positions. I found it best to lean the two headers toward each other so their tops touch and add a bead of super glue along this junction to form a sort of roof over the socket. I tried using another 28 pin socket as a plug. Although the pins lined up, of course, the body shape was such that it did not firmly seat into the original socket. I also tried to quickly solder wires to this second socket but the plastic melted before anything else. I am open to better ideas that still allow me to replace the original FoxDelta PIC if I ever choose to do so. The digital connections to the AD9851 DDS can probably tolerate a little contact resistance but the analog connection to the AD8307 must be as clean and stable over time as possible. I open my case occasionally and pull out this lead and push it back in a few times and then redo the calibration step to insure best possible performance.

During experiments I could see it was going to be important to adjust the Slew and Step rates carefully and adjust them from time to time. Rather than require changing constants in the program to accommodate this, I added two potentiometers to the design, R15 and R16. Suggestions for setting these properly will be discussed later.

The coax relay is not absolutely required for this project but it really makes the controller completely hands-off to operate. There are lots of options here, and several always seem to be available in the used marketplace. The only requirement is that you can find a way to control it with a digital output pin on the Arduino, such that a High switches the antenna to the controller, and a Low switches it back to the radio. The relay I found draws about 100 mA @ 13.8 V and generated 120 volts of back EMF. See the circuitry surrounding Q5 for my solution.

2. Checkout

First a few notes of caution:

1. *Never power the Arduino from the separate 5 V supply and the USB at the same time.* One or the other supply will inevitably be higher, causing current to flow in the reverse direction of the other supply.

2. Although the auto tuner will work fine capturing serial commands from the radio to the computer, *you must disconnect the radio connection temporarily while programming the Arduino*.

For the following, I will assume you have flashed the sample code[x] in your Arduino with the USB connection, built the circuit as per the schematic and connected 5 V to only the FoxDelta. At this point you can connect the antenna Bias-T but do not connect the motor 12 V.

Upon power up, you will first see all four LEDs light up for a second. This is homage to the tradition of testing all lamps. Then you will see the green ComOK light slowly flashing. It will remain flashing like this until receipt of the first successful frequency report from the KX3. Ignore it for now.

Hold the Down switch for a few seconds and release. The yellow Down LED should come on immediately then the red Limit LED should come on after a second or so because there is no current flowing to the motor. With an oscilloscope probe on Arduino pin 11 and the switch down, you should see a 5 V square wave with frequency 490 Hz and duty cycle that varies from 0 to 100% as the Slew pot is adjusted. Repeat for the Up side except the blue Up LED should come on and measure pin 10.

Now connect 12V to the H bridge. Repeat the above and measure the high side of the motor connection. It will be similar but it will be higher voltage and more triangular reflecting a large time constant due to motor inductance.

If this works move on to the KX3 communications. Connect the Ring from the KX3 ACC1 connection and the sleeve to ground. The tip is not used. My KX3 is normally connected to a computer while in the shack so I inserted a stereo “Y” connector to gain access. Power up the KX3

and check that menu entry AUTO INF is set to ANT CTRL. A slight turn of the main radio knob should cause the green ComOK LED to blink indicating a valid frequency report was captured.

3. Calibration

Connect the antenna through the coax relay and the Bias-T. Hold the Down switch until the red Limit LED comes on. Set the Slew pot to mid range. Note the time and hold the Up switch. The Limit LED will go out then come again many seconds later after the motor has rotated from one limit to the next. You want to set the Slew pot so this total travel time is about 20 seconds. Going down is always a few percent faster than going up because the anti-backlash spring is aiding the motor but the difference is not enough to worry about.

The next step is to calibrate the mismatch value. Remove the antenna connection from the Fox BNC connector. While pressing the Tune switch, tap the Arduino Reset switch. After a few seconds you will see a steady rolling patten across all the LEDs. Release the Tune switch and the pattern should reverse and go somewhat faster. After a few seconds the LEDs go out. This indicates you have successfully calibrated the open circuit mismatch value. Using the Arduino IDE Serial Monitor during this procedure, you will see each measured value reported and the final computed value of BAD_MATCH. On my configuration the value is near 400 or a little lower. It's a good idea to perform the procedure a few times and confirm it repeats to within a few percent. If the value varies a lot, you probably have a poor connection to the Fox Delta PIC socket pin 2. The value is stored in EEPROM so it will remain during Arduino power cycles and resets. If for some reason the EEPROM fails to confirm the new value, you will see error code two be

reported and you should retry the procedure. If the EEPROM continues to fail, you have a bad Atmel chip and should replace it.

Now to set the Step pot. This is a little more tricky. Too high and it will skip passed the best match and continue to the limit in that direction. Too low and it will take a long time to find the best match. After the fast slewing step, you want it to find the first best match candidate in 5-10 seconds, turn around and find it again in 3-5 seconds then finally settle on the best position in another few seconds. During this procedure you will see the yellow and blue lights alternate as the algorithm hunts either side of the best match.

The best way to set the Step pot is to use the IDE Serial Monitor to watch the printed report. Tap the Tune switch. On the Serial Monitor you will see two lines that begin with Slewing followed by several lines that begin with Stepping. The Slewing lines are reporting each reading during the initial fast scan. Smaller numbers are better matches. As soon as a significant dip is found, the values that qualified as a dip are reported. The center few of these should be well below the value of BAD_MATCH found during calibration. The Stepping lines reports the finer moves up and down hunting for the best match.

4. Operation

After completing the checkout and calibration, you should have a very good idea how to use the tuner:

1. Power up everything

2. Change rig frequency to get the first frequency report, as indicated by a ComOK LED blink. The tuner will never attempt to move the antenna before the first successful rig report. Automatic tuning is not enabled until the Tune switch is used at least one time.

3. Tune around as desired. You will see the ComOK LED blink each time the rig reports a frequency. Auto tuning will never commence as long you are changing frequency. At some point stop changing frequency. If you have moved sufficiently far away from the last time the antenna was tuned a search will commence automatically. If the antenna was has never been matched since power up, the tuner will start in a random direction but is smart enough to turn around at the end-of-travel if it guessed wrong.

4. Automatic tuning can be interrupted for any reason at any time by tapping any switch. After doing so, automatic tuning will be disabled until you tap the Tune switch again.

5. If the fast slewing step fails to stop and bounces off both limits, try redoing the calibration of the BAD_MATCH value. Also make sure there is a good connection with the Fox Delta PIC pin 2.

6. If the slower fine stepping phase of the search goes on too long, longer than, say, 30 seconds, tap the Tune switch to stop the algorithm. Now listen to the audio from the radio and press the same Up or Down switch that matches the LED motion that was flashing. Hold the switch for a few seconds and release. If you hear a brief increase in the audio level, you know the match was not yet encountered and you should increase the Step pot a little. If you can hold the Up or Down switch a long time and never hear the audio level increase, you know the step jumped over the best match so decrease the Step pot a little.

5. Error Codes

If the tuner encounters an error during operation, it flashes all LEDs a certain number of times to give a clue to the problem. The error code is repeated four times. The codes are as follows:

1. The transmitter is on when a tune attempt was requested.
2. The EEPROM failed to verify the written value.
3. Radio frequency is still unknown when a tune attempt was requested.
4. Radio frequency is too low for the antenna.
5. Radio frequency is too high for the antenna.

6. Easter Eggs

There are a few more features available which you may find useful.

1. **Zero Beat.** During the Tune procedure the DDS frequency may not exactly match the radio frequency, which can cause an annoying heterodyne if there is any leakage through the coax relay from the DDS to the radio. To eliminate this you can calibrate the DDS to zero beat with the radio as follows:

    1. While holding the Up switch tap the Arduino reset. After a few seconds you will see the green ComOK LED flashing. Spin the radio knob a little to force it to report frequency. When a frequency report has been received successfully, the ComOK LED

will go out and the blue Up LED will flash. At this point adjust the Step pot up and down and confirm you can hear the DDS heterodyne. Tune for zero beat and finally release the Up switch. The offset is stored in EEPROM so will remain effective across Arduino power cycles and resets until the procedure is repeated.

2. Note this only provides one zero beat setting. It can be set to work with USB/LSB or CW but not both due to the side tone offset employed for CW.

2. **Frequency Generator:** The FoxDelta can still be used as a general purpose frequency generator. To enter this mode, tap both Up and Down switches at the same time. When successful both Up and Down LEDs will light at the same time. Now the Slew pot is a course frequency control and the Step pot is a fine control. The resolution is about 15 Hz. The coax relay will connect the antenna to the FoxDelta during this mode but of course you can ignore that and connect anything you want directly to the FoxDelta. The frequency is reported in the Serial Monitor if you have that visible. Exit this mode by tapping any switch.

3. Frequency Sweep: Hold the Up switch and tap the Tune switch to start a sweep over the entire auto tuner range. The red Limit LED will flash while the sweep is in progress. A table of frequency and mismatch values is printed in the Serial Monitor window. During the sweep the antenna is switched to the FoxDelta. The sweep is finished when the red Limit LED stops flashing. The sweep can be aborted at any time by tapping any switch.

3. Conclusion

I have enjoyed using the loop tuner now for several months now and find it meets all the requirements quite well. Once in a while it will miss a match but I just tap the Tune switch and it recovers shortly.

Finding a match after a band change typically takes 30-40 seconds, tweaking up after a modest QSY takes 10-20 seconds. Perhaps I could beat these times a little by hand, but now there's no more remembering to first adjust power level; pressing Tune on the radio; jockeying several lights and switches; manually dealing with end-of-travel; adjusting for overshoot; fighting the temptation to tweak it just a little better; fretting over my finals; or QRMing a QSO. Now I just ignore the antenna. I tune the radio around at will and when I find something I want to listen to I stop. If it is far enough away from the last tuning frequency to matter, the antenna is tweaked for the new frequency in a short while and I'm good to go with no effort at all. It feels nearly as though I have a nice broadband antenna. How cool is that?

73, Elwood, WBØOEW

[i] http://www.mfjenterprises.com/Product.php?productid=MFJ-1788

[ii] http://www.elecraft.com/KX3/kx3.htm

[iii] for example see *Ardunio for Ham Radio* by Glen Popiel

[iv] available from *http://arduino.cc/en/Main/Software*.

[v] http://www.analog.com/media/en/technical-documentation/data-sheets/AD8307.pdf

[vi] http://www.analog.com/media/en/technical-documentation/data-sheets/AD9851.pdf

[vii] http://www.foxdelta.com/products/aaz-0914a.htm

[viii] (*insert reference here to ARRL in-depth location for schematic)*

[ix] http://www.mfjenterprises.com/Product.php?productid=MFJ-4116

[x] *(insert reference here to ARRL in-depth location of source code)*