

Drake FS-4 Replacement for the R4 and SPR-4

Doug Crompton, WA3DSP

The Drake R4 and SPR-4 radios depend on a bank of crystals each covering 500 kHz of HF spectrum. Since these radios cover 1-30 MHz (.15 – 30 MHz for the SPR-4) it would require as many as 60 crystals to cover the entire spectrum. Neither radio has enough crystal sockets to do that and even in small quantities these crystals are almost impossible to get. Drake produced a synthesizer to produce all the necessary crystal frequencies called the FS-4. Like all Drake products it is long out of production and a very rare find.



Fortunately we now have several synthesizer chips available making it very cost effective to build a replacement FS-4. The Si5351, a very popular synthesizer chip mounted on a PC card not much bigger than your thumbnail can be purchased for about \$7. The Si5351 has three independent outputs one of which can be used to replace the crystals in your R4 or SPR-4. It can supply frequencies from the kHz range to 200 MHz but for the Drake radios 12-40 MHz is typically used.

The synthesizer chip must be controlled by a computer chip and the Arduino is a popular and very inexpensive choice for this. The interface is I2C which is a type of serial data bus that uses two GPIO lines between the computer and the synthesizer. This makes it very easy to wire. The I2C bus can support many parallel connected devices over the single two wires as each device only responds to its assigned address.

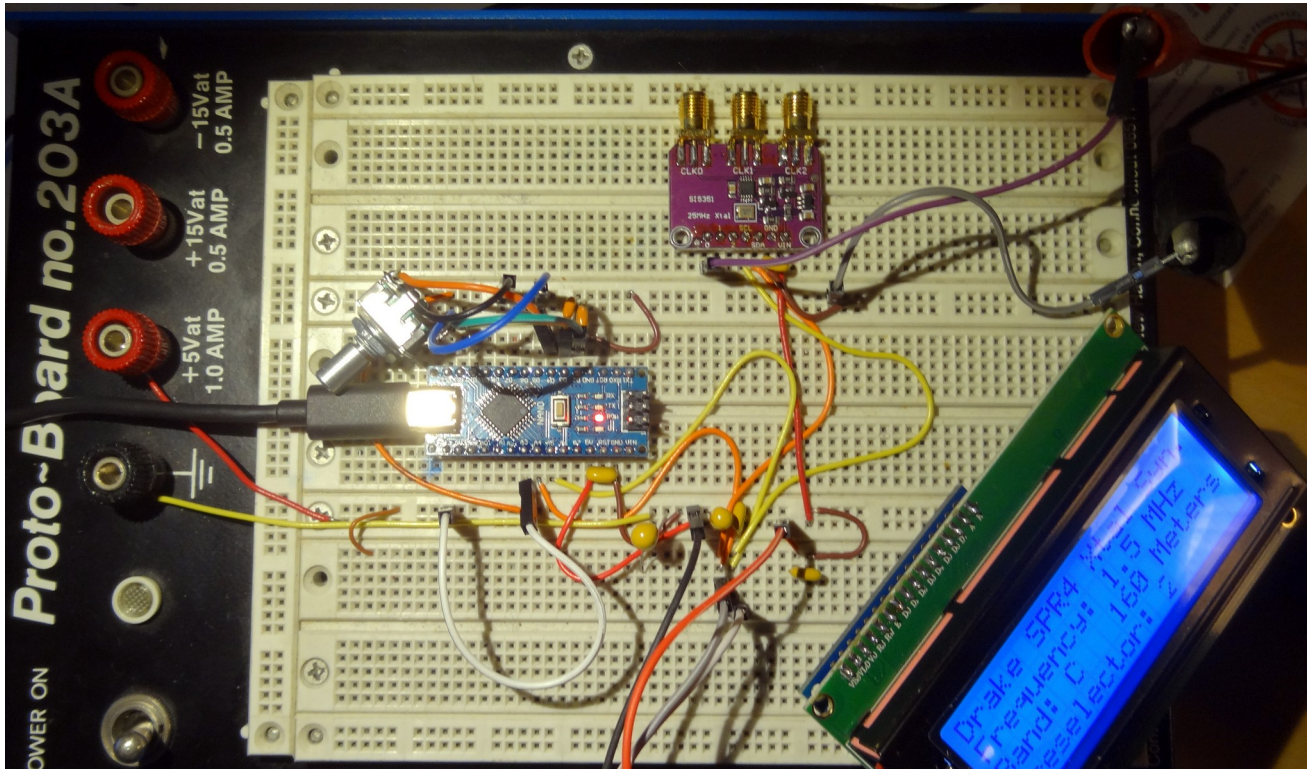
In order to display the current selection to the user a twenty character four line LCD display is used which also is connected to the I2C bus.

The current pseudo crystal is selected by a rotary encoder allowing up/down frequency selection. The encoder also can be pushed in to select predefined quick band changes such as the amateur or SW bands.

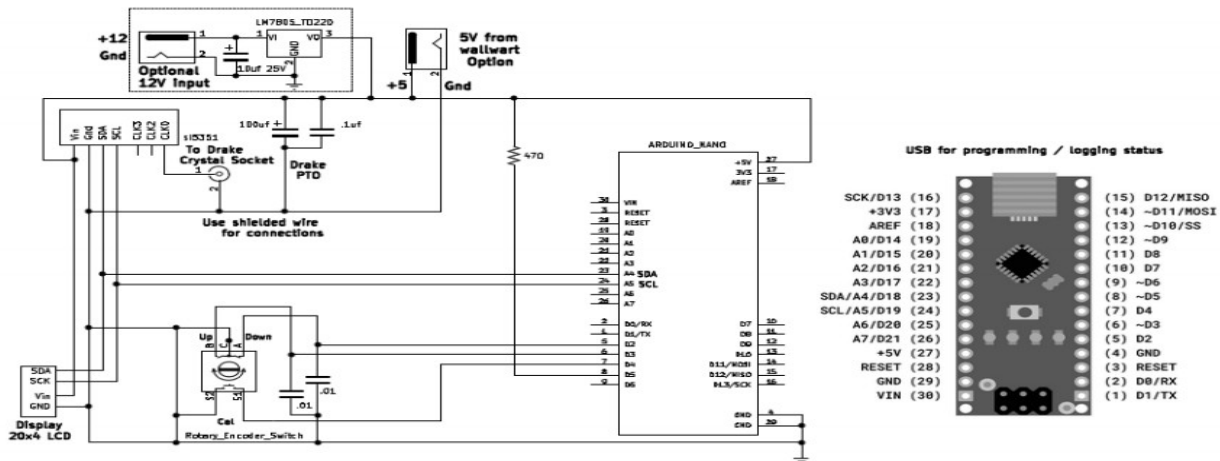
My design uses the original WB6OGD design with several changes. I do not support DDS, OLED displays, or PTO frequency readout. I do use a better rotary.h software library and a very inexpensive Bourns rotary encoder that tracks very well. I also cleaned the code up so hopefully it will be easier to read and understand. Everything that is displayed on the LCD is also output on serial via the USB. So you could run this without the LCD and use the IDE serial monitor for testing.

Building the Synthesizer

The electronic construction of the synthesizer is actually quite simple. There are few connections and all point to point wiring. I always like to bread board a project before I finalize the design. It is also a good check of your hardware before you assemble it in the final enclosure. Every shack should have a proto board.

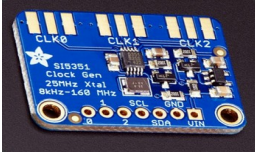


This looks like haywire but it is really quite easy to troubleshoot and test using a proto board as shown here. In this example the Arduino is being powered by the USB connection but it would also run independently using the 5V supplied by the proto board power supply.

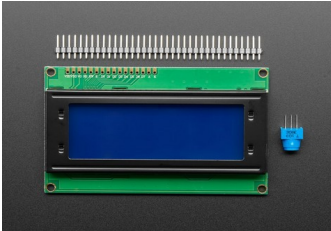


Schematic of the project showing the three main components, the Si5351, 20x4 LCD, and the Arduino Nano. A full size schematic can be downloaded in a link at the end of this article.

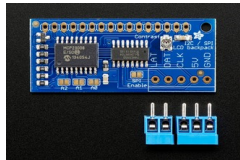
Electronic Parts Requirements



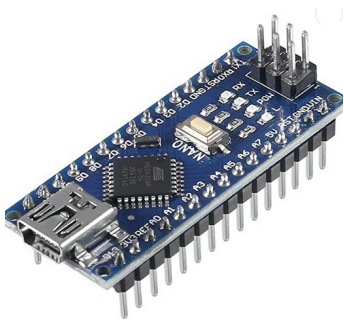
Si5351 Synthesizer board - <https://www.adafruit.com/product/2045> \$7.95



20x4 LCD White on Blue - <https://www.adafruit.com/product/198> \$17.95



LCD Character Backpack <https://www.adafruit.com/product/292> \$9.95
Converts parallel LCD to I2C



Arduino Nano - <https://www.amazon.com/dp/B09PMR9VXP> \$24.95
for four – could also be bought singly. Note this board uses the “ATmega328P (Old Bootloader)” processor selection in the IDE and also uses a USB C connector.



Rotary Encoder - \$1.87

<https://www.newark.com/bourns/pec11r-4215f-s0024/incremental-encoder-2ch-24pulse/dp/46Y2635>

I like to buy and suggest parts from known suppliers that also have good support. For instance Amazon has a no questions asked return policy and I already tested Adafruit as I had purchased an LCD display that had a couple of bad segments. I sent them a photo and they sent a new display and backpack because I had already soldered it on free of charge. Adafruit also fully supports their products with documentation and they have numerous libraries for the Arduino. Here is the link for assembling the display and backpack.

<https://learn.adafruit.com/i2c-spi-lcd-backpack>

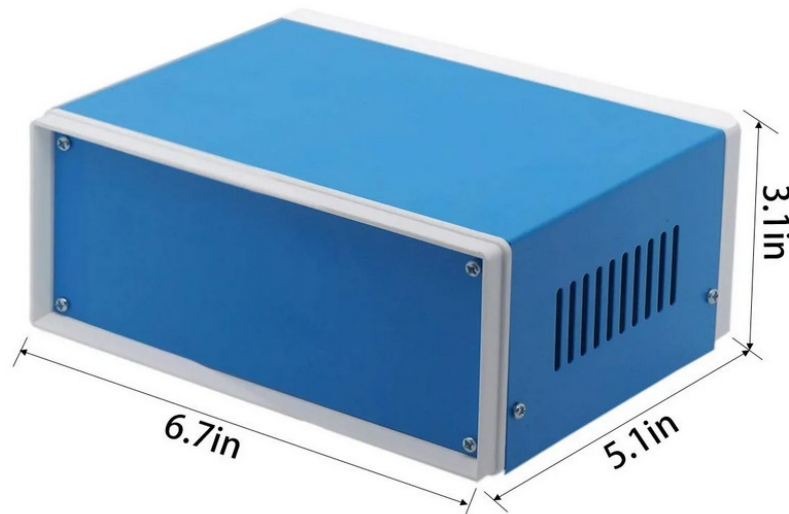
Once assembled the only connections to the LCD are +5V, Ground, and I2C Data, and Clock.

OLED vs LCD

This project originally supported both an OLED and a LCD display as options in the compiling. While OLED displays are clear most are very small and not easy for old eyes to read so I decided to standardize on the LCD display.

Mechanical Parts

How you assemble this into a chassis is up to you as there are many options. Here is what I did. Decorative circuit boxes are hard to find and often very expensive. It is also hard to find the size you need. After checking many places I found a reasonably priced chassis of just the right size on Amazon. I also ordered bezel cutouts for the 20x4 LCD display that fit nicely onto the front panel of this chassis.



Zulkit Electronic Enclosure - <https://www.amazon.com/dp/B08BS243RC> - \$17.99

Cutting an exact and clean rectangular hole in the front panel is difficult for most to do. Using a cutout bezel solves this problem. You cut the rectangular hole slightly larger than the bezel cutout and no one notices that the front panel cutout is uneven. Then the bezel, protective covering, front panel and LCD are mounted using four 4-40 screws. There are a number of places to get bezels, just make sure it fits the 20x4 LCD. Most are a standard size so this should not be a problem. If you have access to a 3D printer there are also plans out there to print them yourself. Here is a Tindie source I used.



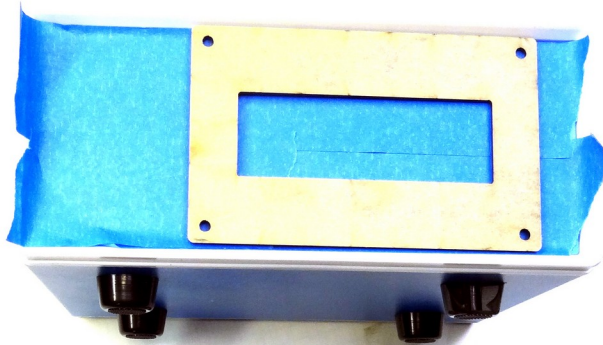
20x4 LCD Bezel with Clear Backplate

<https://www.tindie.com/products/widgeneering/20x4-lcd-bezel-with-clear-backplate/>

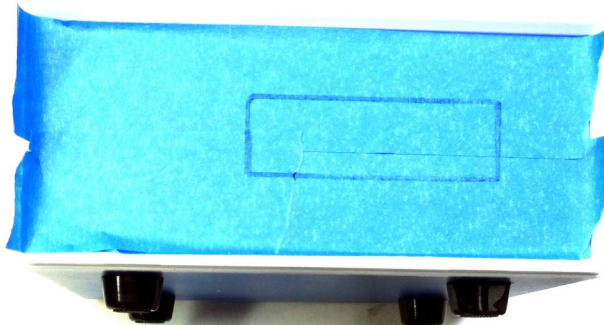
\$8.99 + shipping

Preparing and assembling the Cabinet

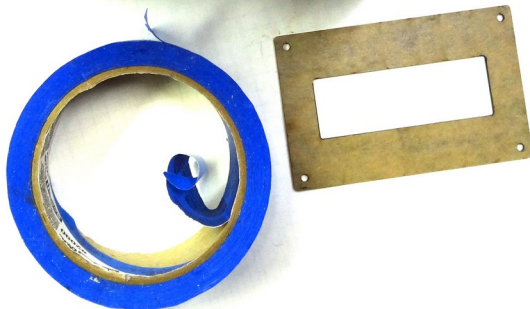
If you are using the cabinet shown above here is how I prepared it.

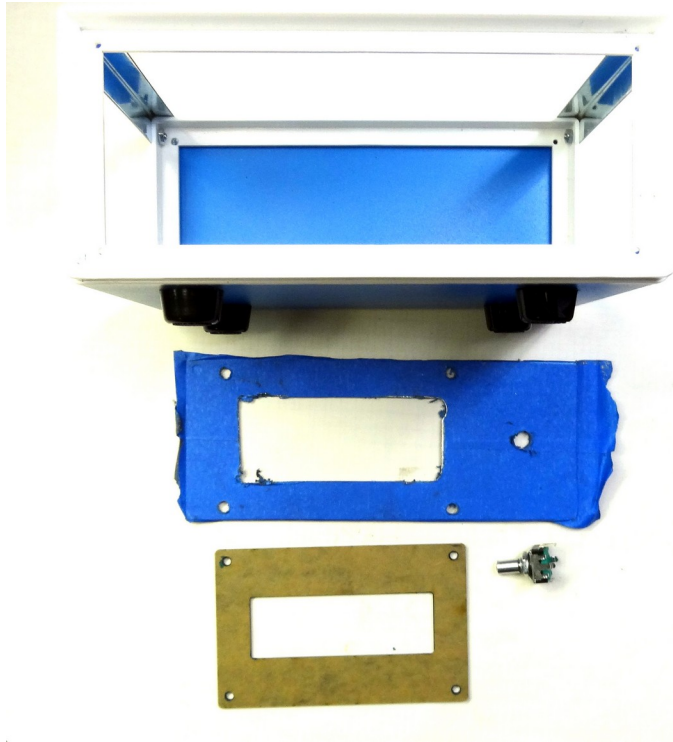


After removing the front panel use painters tape to completely cover both sides. This protects the panel while you prepare it. Then position the bezel and mark the inside cutout for the LCD. You will actually cutout about a quarter inch beyond the markings in all directions so the bezel covers the irregular cutout.



This is what it will look like after marking the cutout. Be sure to position the bezel offset to the right (or left if you prefer) to leave room for the encoder.

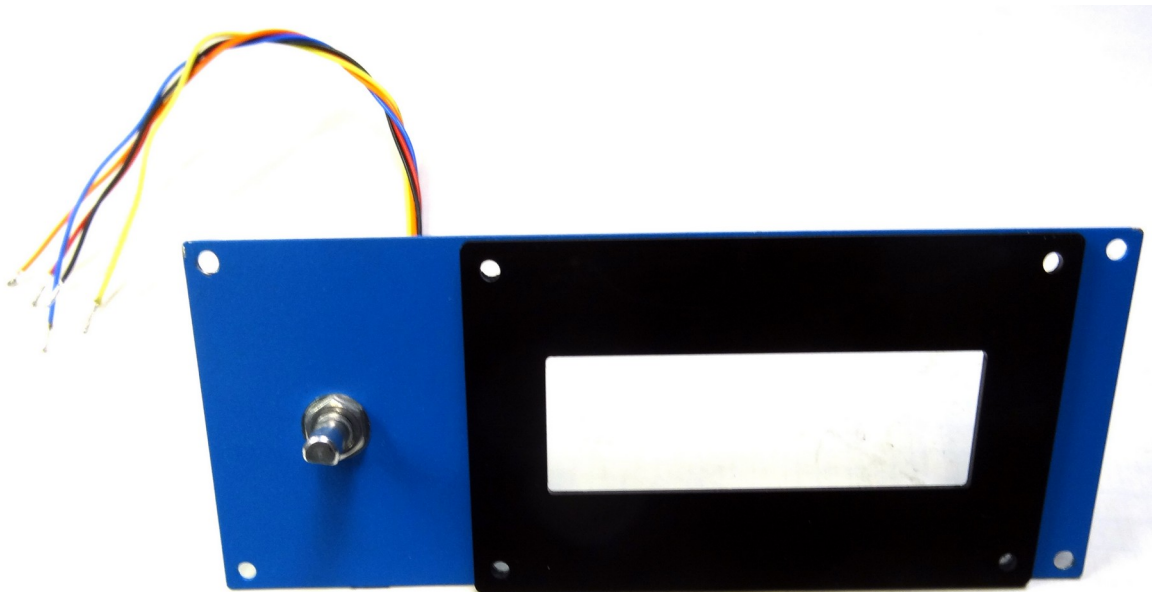




Use whatever method works for you to cutout the rectangle. It is cutout beyond the marked line by at least a quarter inch. Position the bezel over the opening adjusting so the bezel edge to cutout edge matches left to right. The bezel fits completely in the top and bottom direction so no adjustment should be needed. Make sure you have removed enough material so the edges do not show with the bezel installed. Mark and drill the four mounting holes and the hole for the encoder. Clean any edges and remove the painters tape.

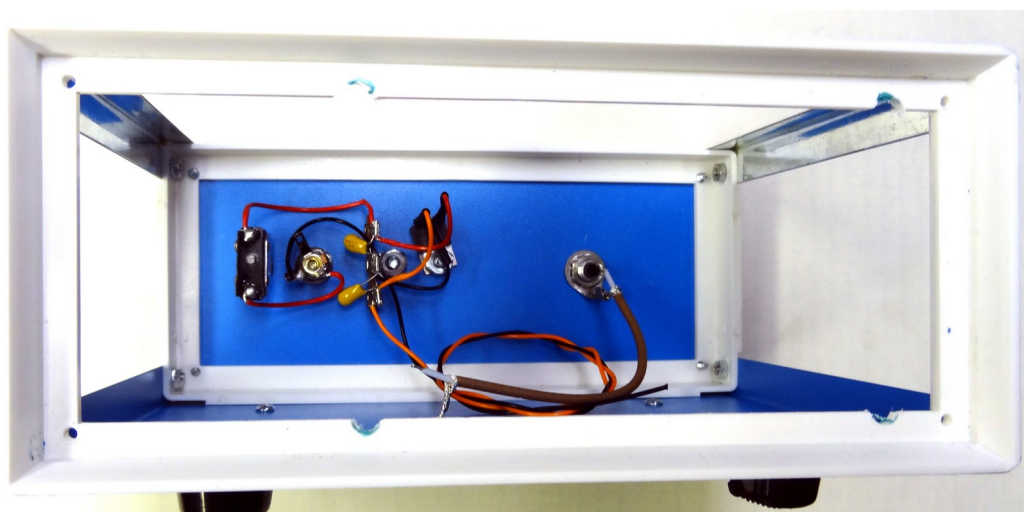


Prepare the wiring to the encoder. This requires 5 wires. Up, Ground, Down, and two wires to the button switch. Colors are not important as long as you identify what they are.



The completed front panel ready for LCD mounting, Use four 4-40, 1 inch screws to mount. The clear PVC that comes with the bezel goes behind the front panel to protect the LCD.

Place the front panel in the frame with the bezel over the front panel so the screw holes line up. Use a marker thru the holes to mark on the frame rail.



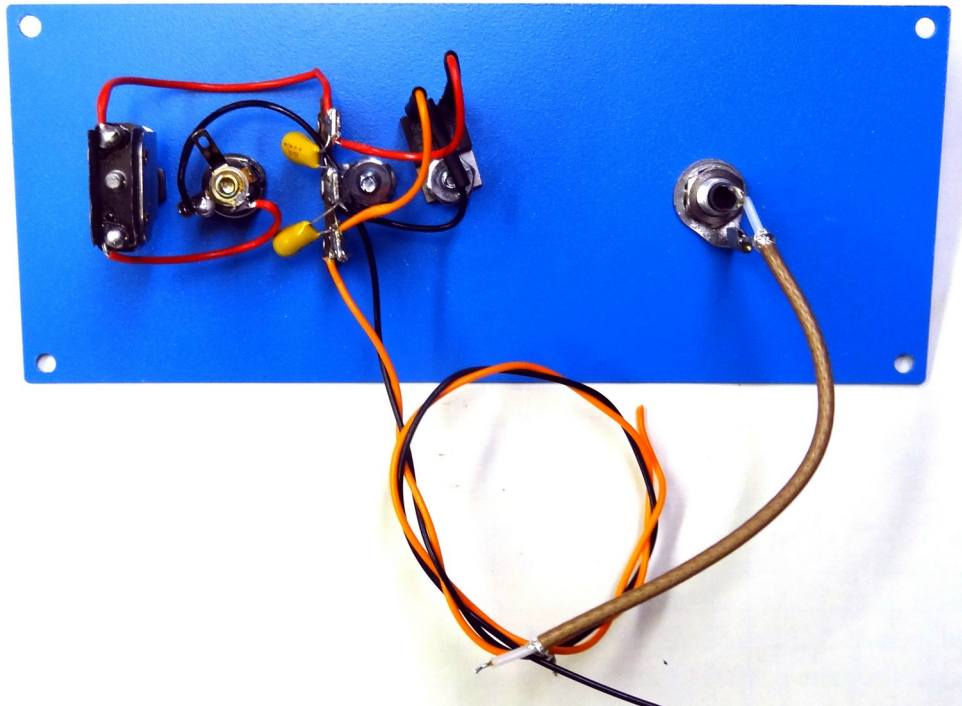
This photo shows where the rail was relieved so the mounting screws could go through. You could carefully drill through the front panel and bezel, or mark the position and use a file or Dremel tool.

Preparing the Rear Panel

The rear panel has a coaxial power connector, a power switch, and an RCA connector for the output. There are two options for supplying power, either 12V or 5V. If you use the 5V option it must be clean power and well regulated at 5V. It could come from a wall wart such as one used for the Raspberry Pi or from a USB connection to an external device. I decided that most shacks have 12V power available so I used a 7805 type regulator mounted on the rear panel. Both of these options are shown on the schematic. When using the 12V option the input voltage can be anything from 9 to 30 volts but more typically 13.8V from a shack power supply or 12V from a wall wart. Current requirements are less than 1/2A. 12V wall warts are readily available.



The rear panel showing the RCA output connector, the coaxial power input, and the power switch.



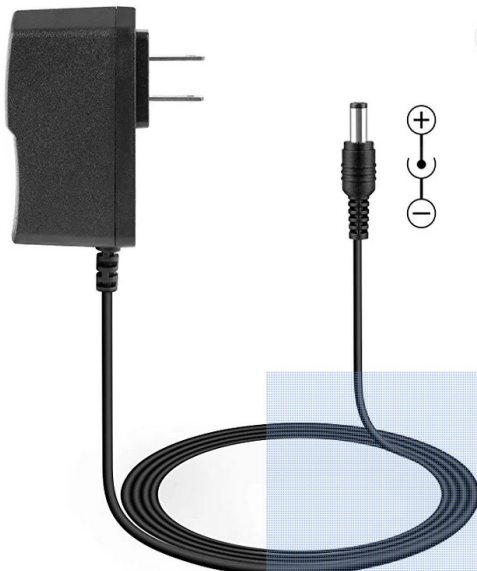
Inside of rear panel showing power switch, coaxial power input, 5V regulator, and output connector.

Two power supply options are shown below or wire your own jack to an existing power supply. Make sure to observe polarity! The center pin of the coaxial jack is positive. Use the 12V option only if you are using a 7805 5 volt regulator in your box.



5V wall wart \$5.29

https://www.amazon.com/Adapter-Connector-Wireless-Security-Switches/dp/B091Y7WJTL/ref=sr_1_3



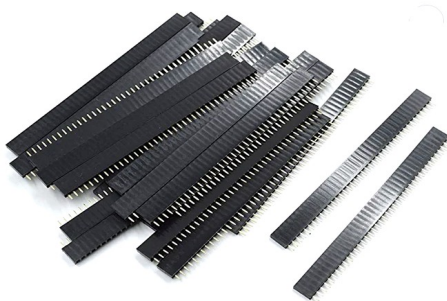
12V wall wart \$8.99

https://www.amazon.com/Adapter-Switching-100-240V-Transformer-Security/dp/B08C53XF9W/ref=sr_1_3

Building the main board

There are many options here. The wiring is not difficult. The Arduino and Si5351 boards generally have plugin pins so using a socket strip on a prototype PC board allows you to easily swap in and out those components.

I cut a portion of a proto board with solder pads to mount the Arduino and the Si5351 sockets. I like to socket these components because although a failure is unlikely if it happened it would be a real task to remove them. This also give you the capability in the case of the Arduino to swap it in and out and program it externally. Here are two examples of socket material which allow you to cut as many pins as you need for a project. Both are overkill quantity wise for this particular job but either would be helpful for future projects.



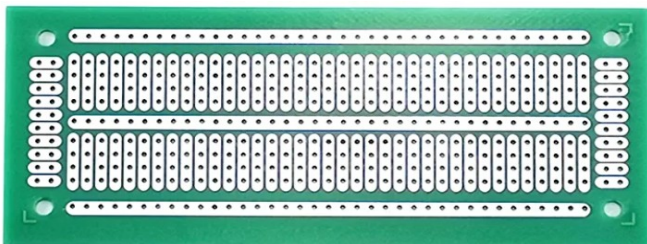
Breakable IC Sockets. Hxchen 1x40 Pin 2.54mm Pitch Straight Single Row PCB Female Headers - (30 Pcs)

\$8.99 <https://www.amazon.com/dp/B07VBYD2C3>



Glarks 120Pcs 2.54mm Straight Single Row PCB Board Female Pin Header Socket Connector Strip Assortment Kit for Arduino Prototype Shield(Single Row)

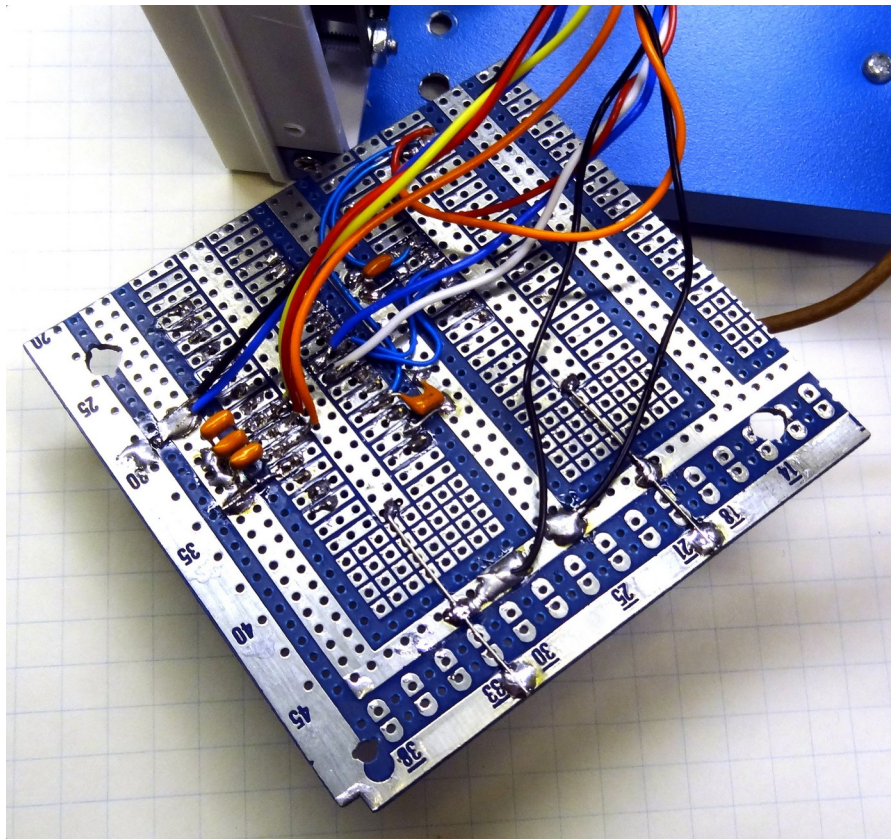
\$12.84 <https://www.amazon.com/dp/B076GZXW3Z>



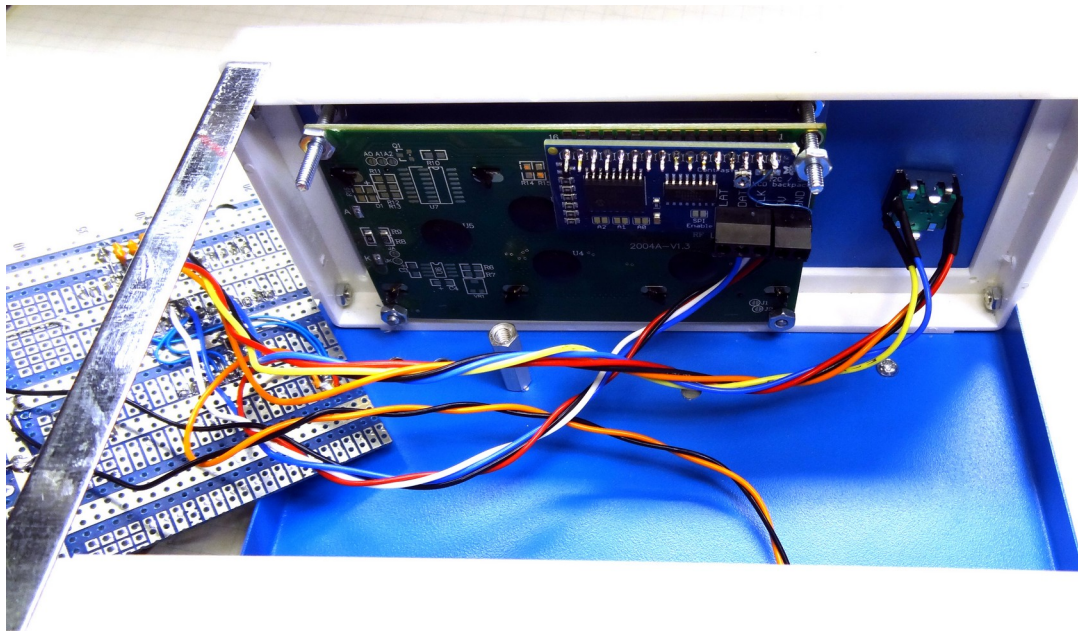
One of many examples at Amazon for a proto board. Nice because it gives many connections per pad.

YUNGUI Printed Circuit Board,47X116MM PCB Strip Breadboard for Electronic Project and DIY Soldering(Pack of 8)

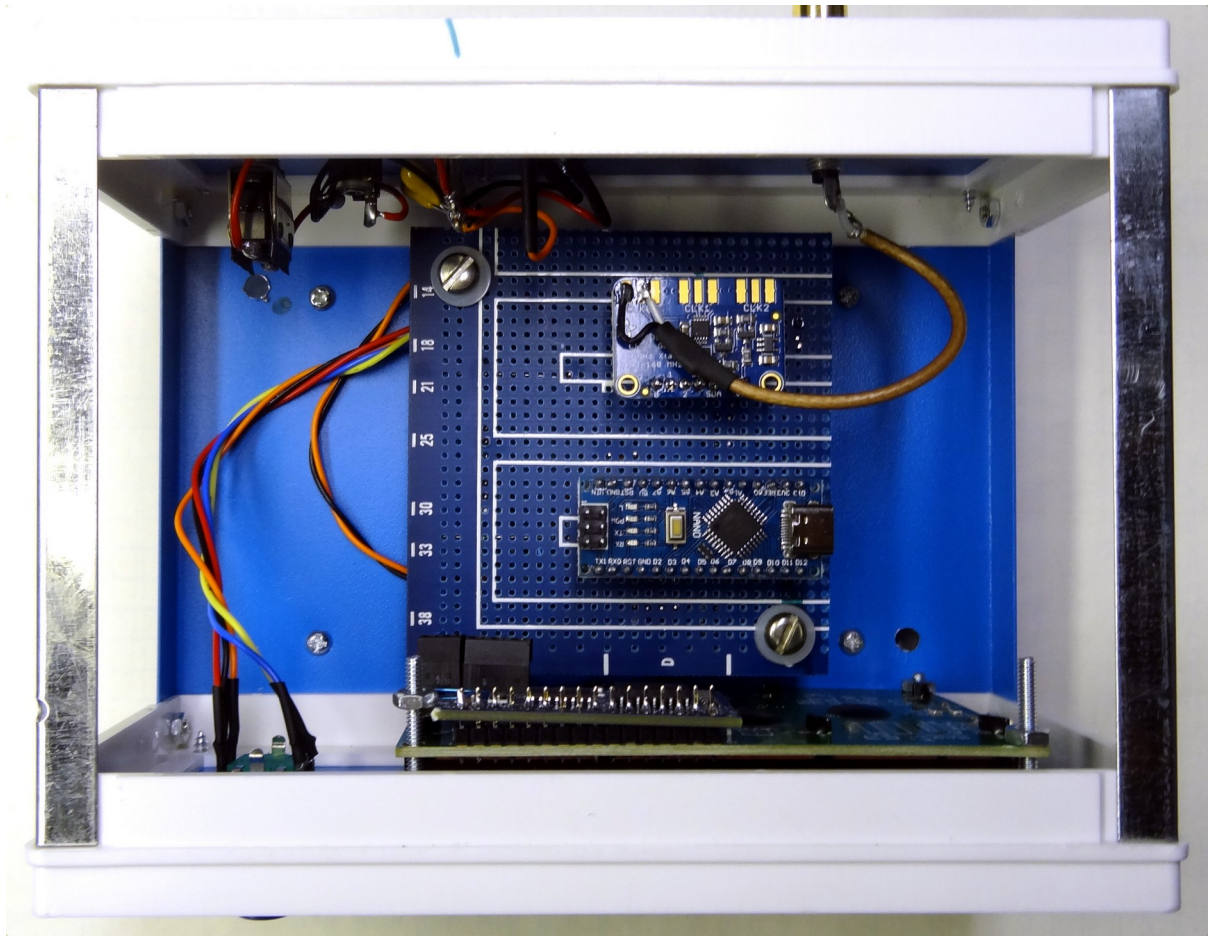
\$11.59 https://www.amazon.com/YUNGUI-47X116MM-Breadboard-Electronic-Soldering/dp/B08SQ34KTC/ref=sr_1_34



Wired circuit board ready to mount in chassis.



Another view before the circuit board is turned over and mounted. The display board is mounted with four 4-40 1.5 inch screws and nuts. The screws go through the bezel and chassis, with the clear PVC behind the front panel and then the display. To make it easier I secured the bezel, front panel and clear PVC with four nuts and then mounted the display and used four more nuts. Cut the screw lengths as needed. DO NOT tighten these too much as the PVC can crack and the display can be damaged.



Circuit board mounted showing Arduino and Si5351. The output of clock 0 is connected via coax to the rear RCA connector. Arduino USB connector faces towards the right side for access to programming.



Completed front panel with knob installed.

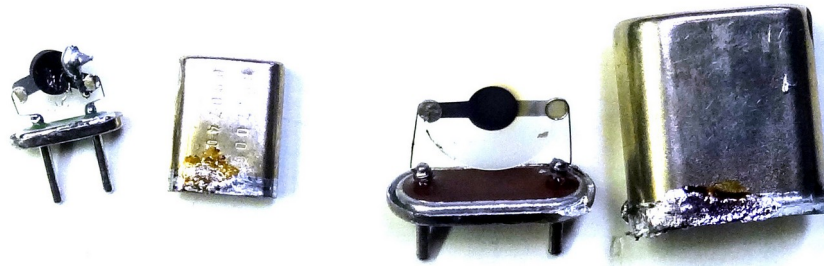


Unit in operation programmed for the SPR-4 with 40 meters selected. The output frequency for this range should be 18.090.000. It is about 12 Hz. low and certainly within the accuracy of the measuring instrument.

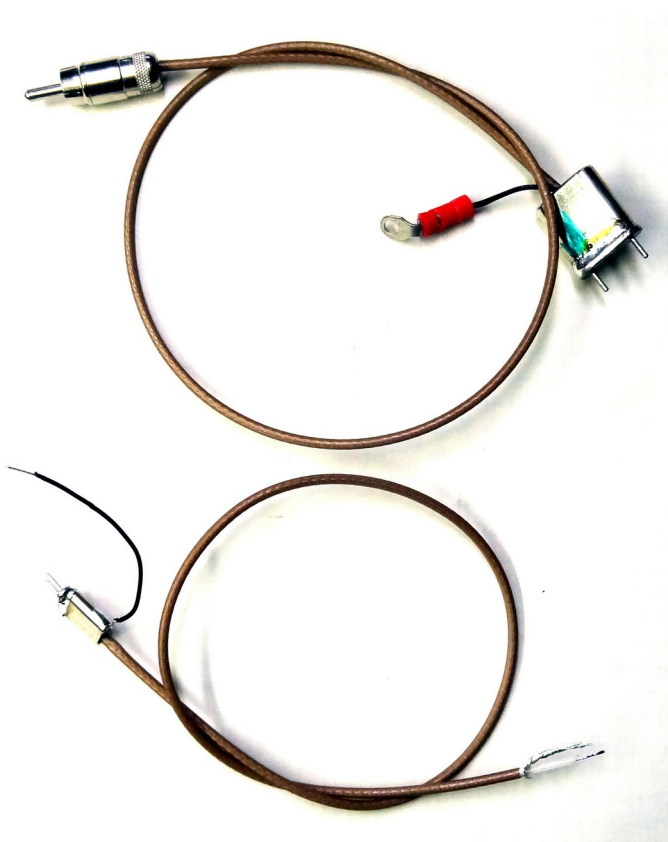
Turning the rotary switch selects the band up or down and pushing the knob in switches to the next quick band change. In the case of the SPR-4 it selects the most popular SWL bands as well as all Amateur bands.

Interfacing to the Radio

Interfacing to the R4 or SPR4 is via an unused crystal socket. An easy way to do that is to take an existing crystal, preferably one you have no use for and modifying it. The R4 uses HC6U crystals and the SPR-4 uses HC25U crystals. The R4 has rear panel mounted crystals so one of those positions could be used for synthesizer input. On the SPR-4 the crystals are internal and a more convenient way to make the external signal available would be to make a cable that plugs into an unused crystal socket and goes to a newly mounted female RCA plug on the rear panel. Then a male RCA to male RCA cable would connect the synthesizer.



The HC25U and HC6U with their covers removed. Grab the pins in a vice and heat the case while you pull it off.



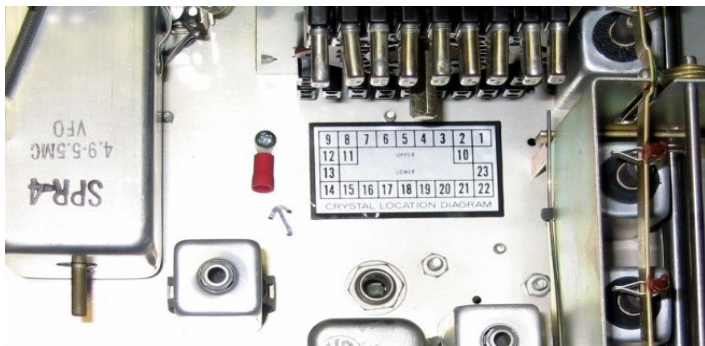
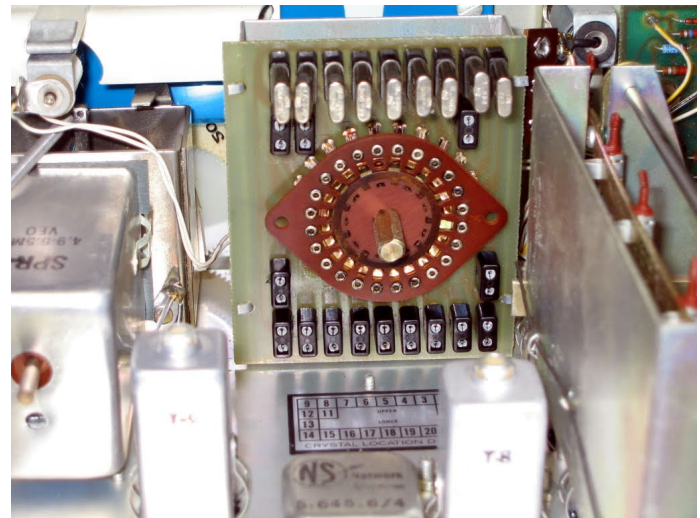
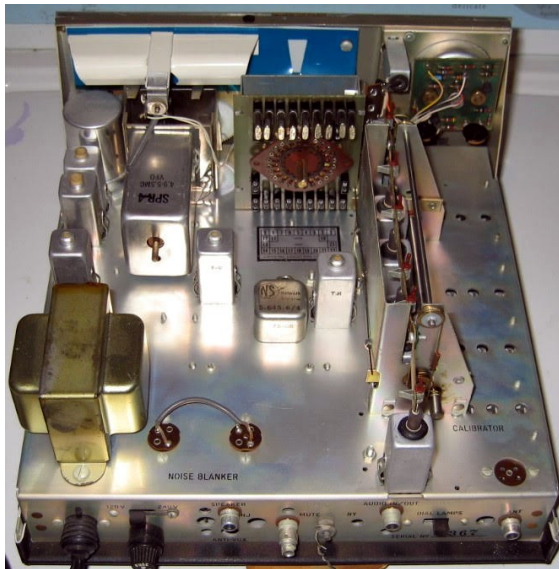
Remove the crystals and using small coax, RG174 or a Teflon equivalent solder the inner conductor to the one of the pins. Solder a short piece of wire to the shield. The other pin remains unconnected. Here showing the HC6U assembly at the top and the HC25U assembly at the bottom.

Both crystal covers have a hole drilled in them to accommodate the coax and the ground lead. For the HC6U I carefully used a Dremel cutting tool with the crystal in a vice.

Mark the crystal case so you know which pin is the inner conductor. This will plug into an unused crystal socket with this pin towards the non-common side of the socket.

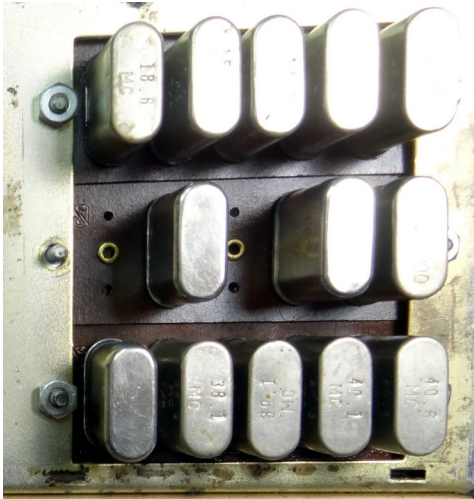
For the R4 (photo top) the cable ends with an RCA connector. For the SPR-4 the leads are tinned to solder to a back panel mounted RCA jack.

SPR-4 Photos



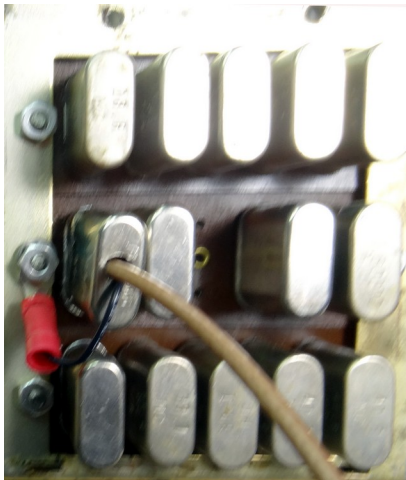
Photos showing the crystal bank in the SPR-4. The “common” side of the crystals face away from the switch. A crystal adapter as shown in the previous page would plug into the one of the unused crystal positions preferably close to the ground lug in the bottom left photo above and the coaxial cable would go through an existing opening or one you make to below chassis to a rear panel mounted solder female RCA jack. An unused rear panel RCA plug or one you install in a spare hole serves to bring the signal in. This is shown in the above right photo just to the right of the speaker connector.

R4 Photos



Rear crystal bank on an R4. The top bank is 1 to 5 (l-r), the middle is 10 to 6 (l-r) and the bottom is 11 to 15 (l-r).

In this example crystal number 10 is vacant. I have also removed the nut to the left of that socket. This is where the ground will go.



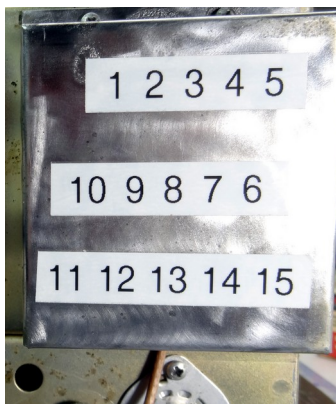
The crystal adapter is plugged into socket 10 and the ground lug with lock washer is secured with the original nut.

The upper and middle banks top pins are common so the coax center conductor for those banks is the lower crystal pin. The bottom bank is the opposite, the bottom pin is common.

The coax bends down and goes in between the bottom row of crystals and out the bottom of the cover.



The bottom of the crystal cover is relieved to accommodate the coax exit, with a piece of tape over the edge to protect the coax.



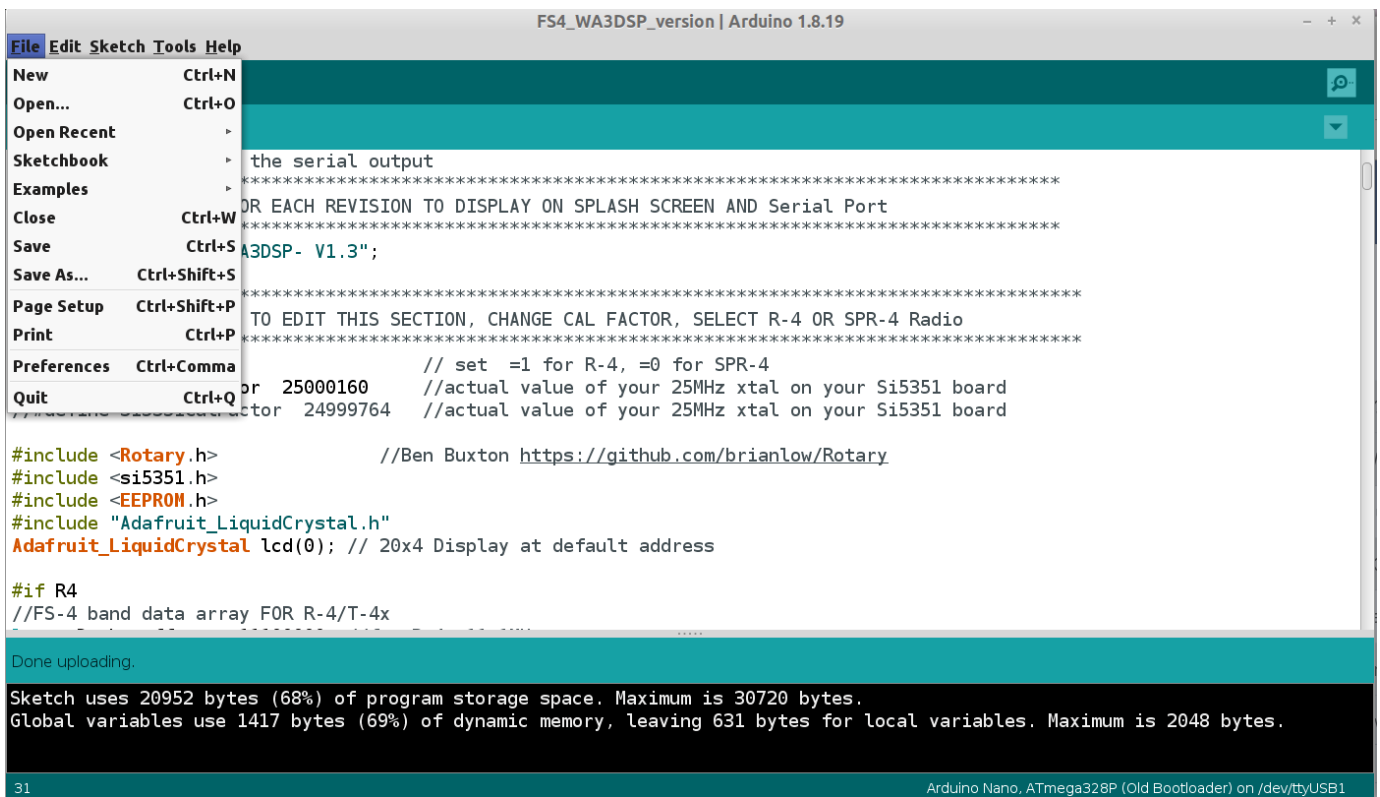
The crystal cover is installed with the coax extending through the bottom. The coax is long enough to reach the synthesizer placed on top of the receiver.

Arduino Sketch and Programming

The Arduino requires programming to perform its desired task. In order to do this you need several things. First the compiling and programming software called the Arduino IDE or integrated development environment. Versions are available for Windows, macOS, and Linux. Download and install the version of your choice. Information and downloads are at -

<https://docs.arduino.cc/software/ide-v1/tutorials/arduino-ide-v1-basics>

This will allow you to load the ‘sketch’ for the FS-4. A “sketch “ is just the source code or user readable code for the program you want the Arduino to perform. Sketches often require “libraries” which allow you to use simple commands in your sketches to control more complex hardware or software routines. My FS-4 sketch requires libraries for the display, the synthesizer chip, EEPROM storage and the rotary encoder. If these libraries are not loaded the sketch will not compile properly. Libraries are defined at the beginning of a sketch as #include statements. Below I will briefly explain the compiling and programming process.

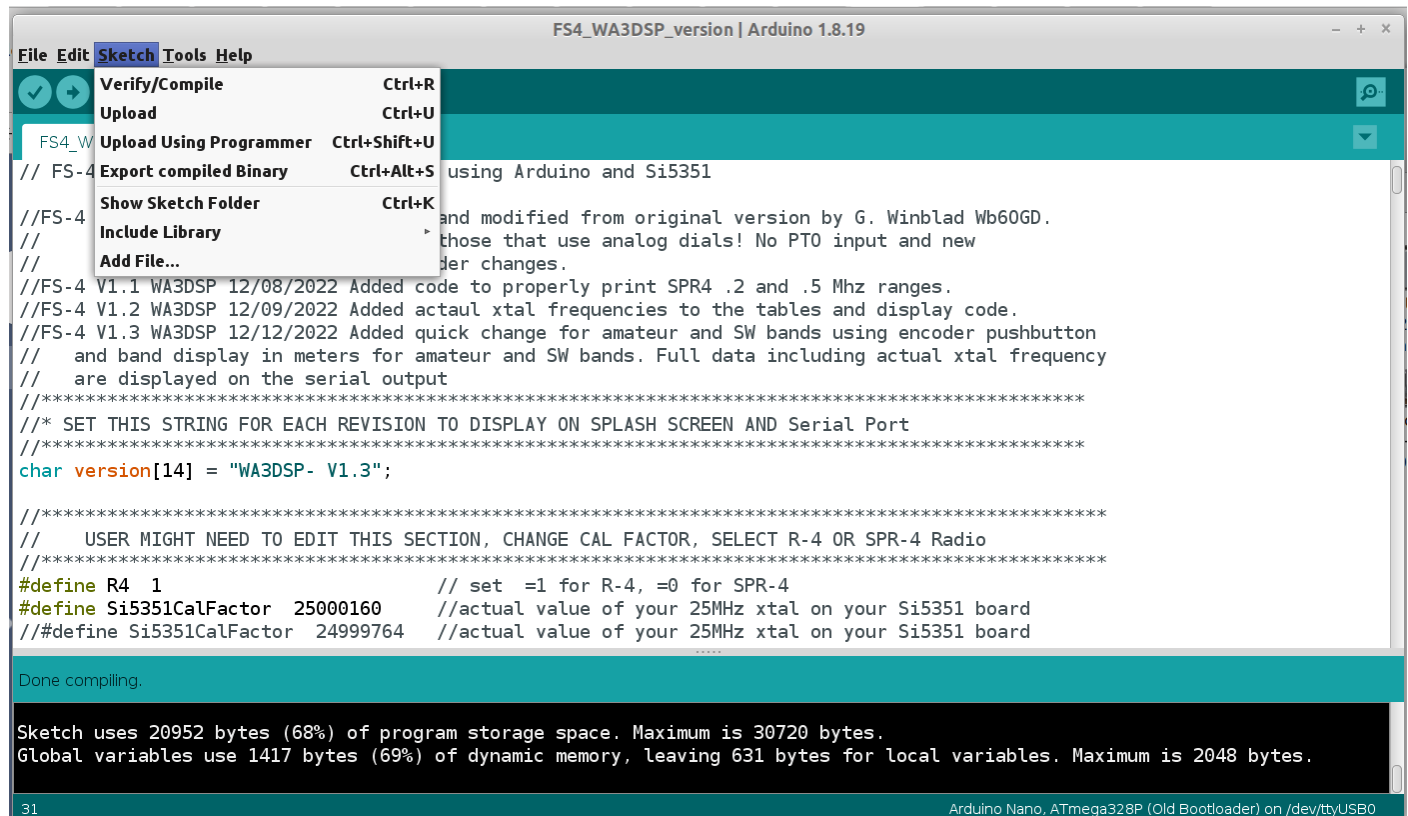


After opening the IDE the first thing you do is open the sketch. These examples show using the IDE version 1 in Linux Ubuntu but it should be similar in all distros and versions. Once you open the FS4 sketch you can make a few changes to it. The only user changes that are necessary are selecting whether it is to be compiled for an R4 or SPR4 and setting the calibration of the Si5351. The lines are near the top of the sketch.

```
*****
// USER MIGHT NEED TO EDIT THIS SECTION, CHANGE CAL FACTOR, SELECT R-4 OR SPR-4 Radio
*****
#define R4 1 // set =1 for R-4, =0 for SPR-4
#define Si5351CalFactor 25000160 //actual value of your 25MHz xtal on your Si5351 board
```

The R4 setting is either a 1 or 0, 1 it is set for R4, and 0 for an SPR-4. The calibration setting sets the 25 Mhz crystal clock frequency of the Si5351. Since every board could be slightly different this gives you the option to set it right on. I found the three boards I bought from Adafruit were all very close to 25 Mhz as you can see by the offset in my example. With the cal factor shown the output frequencies were off by about 12 Hz low and it was not worth messing with it any further. If you have a need to change it you will need to change this cal factor value, and compile, and upload the code each time you change it and check the output crystal frequency for the change.

Nothing else needs changing in the code for operation but if you understand the code you can make changes as you desire. Just be sure to make a backup copy.



```

File Edit Sketch Tools Help
Verify/Compile Ctrl+R
Upload Ctrl+U
FS4_WA3DSP Upload Using Programmer Ctrl+Shift+U
// FS-4 Export compiled Binary Ctrl+Alt+S using Arduino and Si5351
// FS-4 Show Sketch Folder Ctrl+K and modified from original version by G. Winblad Wb60GD.
// Include Library those that use analog dials! No PTO input and new
// Add File... der changes.
//FS-4 V1.1 WA3DSP 12/08/2022 Added code to properly print SPR4 .2 and .5 Mhz ranges.
//FS-4 V1.2 WA3DSP 12/09/2022 Added actual xtal frequencies to the tables and display code.
//FS-4 V1.3 WA3DSP 12/12/2022 Added quick change for amateur and SW bands using encoder pushbutton
// and band display in meters for amateur and SW bands. Full data including actual xtal frequency
// are displayed on the serial output
//*****
//* SET THIS STRING FOR EACH REVISION TO DISPLAY ON SPLASH SCREEN AND Serial Port
//*****
char version[14] = "WA3DSP- V1.3";

//*****
// USER MIGHT NEED TO EDIT THIS SECTION, CHANGE CAL FACTOR, SELECT R-4 OR SPR-4 Radio
//*****
#define R4 1 // set =1 for R-4, =0 for SPR-4
#define Si5351CalFactor 25000160 //actual value of your 25MHz xtal on your Si5351 board
//#define Si5351CalFactor 24999764 //actual value of your 25MHz xtal on your Si5351 board

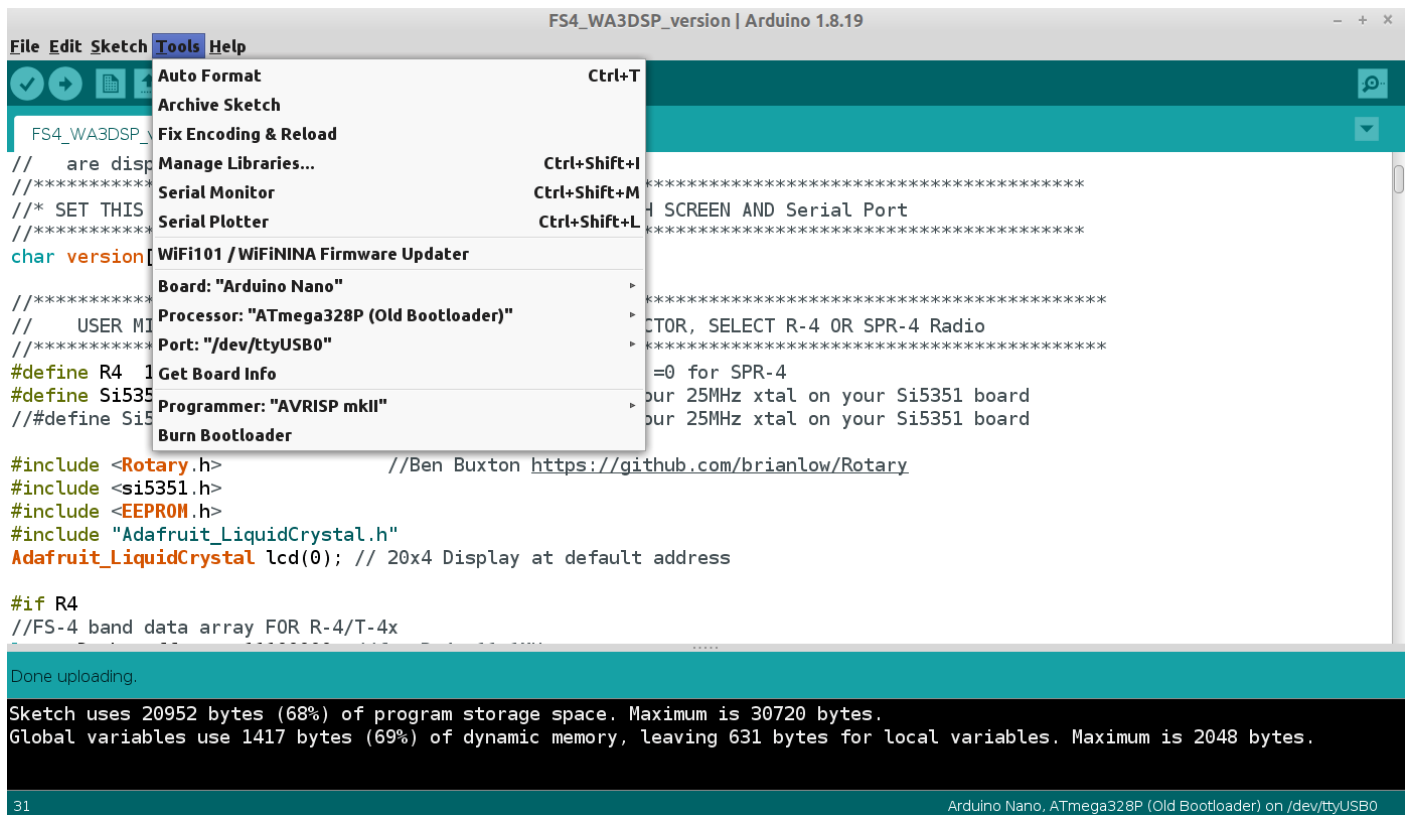
Done compiling.

Sketch uses 20952 bytes (68%) of program storage space. Maximum is 30720 bytes.
Global variables use 1417 bytes (69%) of dynamic memory, leaving 631 bytes for local variables. Maximum is 2048 bytes.

31 Arduino Nano, ATmega328P (Old Bootloader) on /dev/ttyUSB0

```

Now you need to load the libraries needed for the sketch. Under the “Sketch” drop down select “Include Library” and select EEPROM, then again select “Include Library” and scroll down to “Adafruit_LiquidCrystal.h and select. The Si5351 and rotary libraries are supplied in the zip file so all you need to do is again select “Sketch” then “Include Library” and then “Include .zip Library” This will bring up a directory/file selection page. Select the “libraries” directory of the provided FS4 zip file and select Etherkit_Si5351. Then do this again and select MD_REncoder. Return to the *Sketch > Include Library menu*. You should now see the libraries in the drop-down menu. They are ready to be used in your sketch. This should load all of the required libraries. This only needs to be done once, your IDE now knows to use these library files when they are included your sketch. The libraries specified are for the sketch and hardware specified in this document. DO NOT use any other Si5351 or rotary library than those supplied. Both the Adafruit LCD and EEPROM libraries are included in the Arduino IDE but must be loaded.

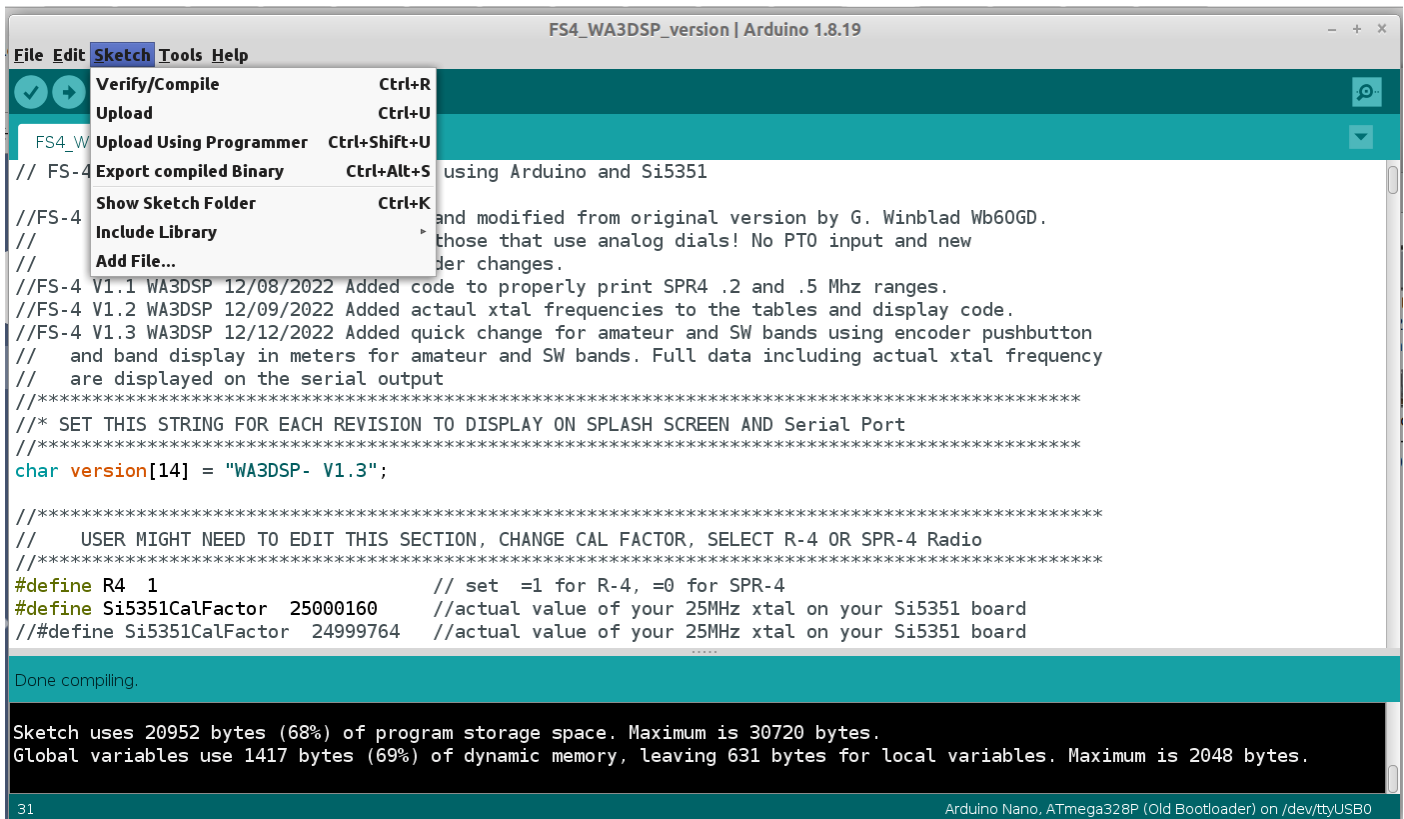


Now we have to tell the IDE what board we are using and what port it is on. For this you need to connect the Nano USB port to the computer before you start the IDE. Then start the IDE and select "Tools" and the port where you connected the Nano should display on the "Port:" line. You can also force a specific port by selecting the "Port:" line and selecting the port. As you can see above the selected port is /dev/ttyUSB0. This is a Linux port, for Windows it would be a COM port.

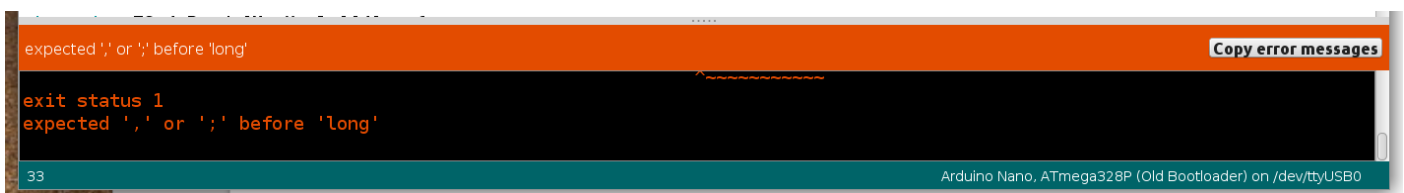
Now again select "Tools" and select the "Board:" line. Select Arduino Nano from the drop down and it should display on the "Board:" line as shown above.

Next click on "Tools" again and select the "Processor:" line. You want either the "ATmega328P" or the "ATmega328P (Old Bootloader)" depending on the board. The ones specified in this document use the "Old Bootloader" but many other boards do not. If you use a Nano board from a different source you may need to use just the "ATmega328P" selection.

No other settings under "Tools" need to be changed. Note the "Serial Monitor" under "Tools." You could use this selection in the testing stages to monitor what the text output. All changes are output on the USB cable connected to the Nano and viewable on the serial monitor set to 9600 baud.



Finally we are ready to compile the code and upload to the Nano. Select the “Sketch” drop down and then “Verify/Compile” The IDE will then compile the sketch. If all goes well you should see the text as show above at the bottom on the screen. If there is a error the text will be written in a red with an orange divider at the top as shown below. Scroll up to the first error and correct that and compile again. Most of the time this will solve the problem but if not again scroll up and check for other problems. Continue to correct things until you get a clean compile.



Now with your Nano USB connected select “Download” and the programming process to the Nano should commence. This only takes 10-15 seconds or so. When complete the bottom display will show “Download Complete.” If there is a error in the programming this will again show in red in the bottom window.

If you experience an error check that you are using the correct processor and boot code, in the case of the Nano the “ATmega328P” or “ATmega328P (Old Bootloader)” If you are not sure which your board requires try one then the other.

Make sure you have the right port and have communications with your Nano. If you are not sure select Tools then Port so it shows all possible connections then disconnect your Nano. One port should go away. That is the communications port and should be the one selected when the Nano is plugged in.

Sometimes I have found that things get locked up for some reason. If you get in a state where it won't program make sure you have saved the sketch, close the IDE, disconnect the Nano USB, wait about 10 seconds, reconnect the Nano, then restart the IDE, and try again.

If the program running in the Nano is outputting on the serial bus it sometimes clashes with the programming. Try multiple times to program and if it does not work reset everything as described in the previous paragraph.

When programming observe the LED's on the Nano. You can determine if it is programming as one LED lights during the process and then goes out.

Programming using Pre-compiled Sketches

If you want to avoid all of the above you can program the Nano with a pre-compiled Sketch for the R4 or SPR-4 which are supplied in the binaries directory of the zip file. This eliminates the need for the IDE or loading any libraries. The downside is that you can't make any changes to the sketch. In general this should work because the only other setting you would make is the Si3531 calibration which is usually pretty close. This also requires getting some things right in the avrdude command line Here is how to do it -

First check to see if you already have the avrdude program. Type avrdude at the command prompt on your computer. If the response is not found and you do not have avrdude download the program. In Linux this would just be - **sudo apt-get install avrdude**

[Installing avrdude in Windows 10](#)

For other OS's see - <https://github.com/avrdudes/avrdude>

Then attach your Nano to your PC and execute the following line with appropriate changes as noted. I would recommend cutting and pasting then making changes to ensure that the syntax is correct.

```
avrdude -C/etc/avrdude.conf -patmega328p -carduino -P/dev/ttyUSB1 -b57600 -D  
-Uflash:w:FS4_WA3DSP_R4.hex:i
```

-C is the location of the avrdude config file.

-P is the USB port the Nano is connected to. For Windows this would be a comm port

The .hex file you want to write is between the "w:" and ":" at the end of the line. Shown is the pre-compiled R4 file. It needs to be in the directory this is run from or a complete path to it. Use FS4_WA3DSP_R4.hex for the R4 and FS4_WA3DSP_SPR4.hex for the SPR-4.

This will write the functional pre-compiled code to your Nano. When complete with no errors disconnect the Nano and use in the project.



R4C tuned to 80 meters using the synthesizer input. Note the XTALS switch is switched to position 10 where the synthesizer is plugged in on the back crystal bank. This particular R4C has all the ham band crystals plus some SW as well so the synthesizer is not really needed in this case unless you want to listen on other SW bands it does not have crystals for. Switching between the NORM position and position 10 resulted in no noticeable difference in signal level or frequency. You can leave the R4's XTALS switch in position 10 and cover the entire HF spectrum by instead using the knob on the synthesizer. The synthesizer display tells you where to set the band and approximate setting for the preselector. Pushing the knob quickly selects all HF ham bands.

Links

[Drake SPR-4 Manual](#)

[Drake R4C Manual](#)

[Drake FS-4 Manual](#)

[Schematic for this Project](#)

[Si5351 Library Manual](#)

[Adafruit LCD Info](#)

[Adafruit Backpack Manual](#)

[Rotary Encoder Library Info](#)

[Arduino Documentation](#)

[Drake Radio groups.io](#)

Thanks to Gary, WB6OGD, Ray, WB2JJO and others on the Drake groups.io site for help in designing and documenting this project. I basically took existing Arduino code and adapted it to my needs using currently (Dec 2022) available hardware. Hopefully this will help others to keep their now “antique” Drakes fully functioning in light of scarce crystals. This same method with code modification could be used for other brand radios. The Si3531 could also be used in a modern HF VFO Design.

I usually monitor the Drake Radio groups.io site but you can also reach me directly at wa3dsp@gmail.com. I would be glad to answer any questions about this project.