Arduino CAT Controller for HPSDR

John Melton, G0ORX/N6LYT
4 Charlwood Close
Copthorne
West Sussex
RH10 3TG
England
john.d.melton@googlemail.com

Abstract

Simple CAT Controller for HPSDR using an Arduino micro-controller and a few switches and a step encoder.

Introduction

One of the perceived problems of the HPSDR project is that the radios do not have any knobs and buttons. PowerSDR has a CAT interface for controlling the radio that uses a serial interface. This interface is used by several add on applications that also require control of the radio, typically using a virtual serial port.

The Arduino micro-controller provides an ideal platform to build a controller with knobs and buttons. The basic Arduino UNO is cheap and provides for a number of analog and digital interfaces. Simply connectng up a few push switches and step encoders allows us to develop a custom controller.

Basic Design

A controller can be implemented using an Auduino UNO. These are readily available from several on-line resources including EBay. In the UK they can be found as cheap as £5 ($7.50).

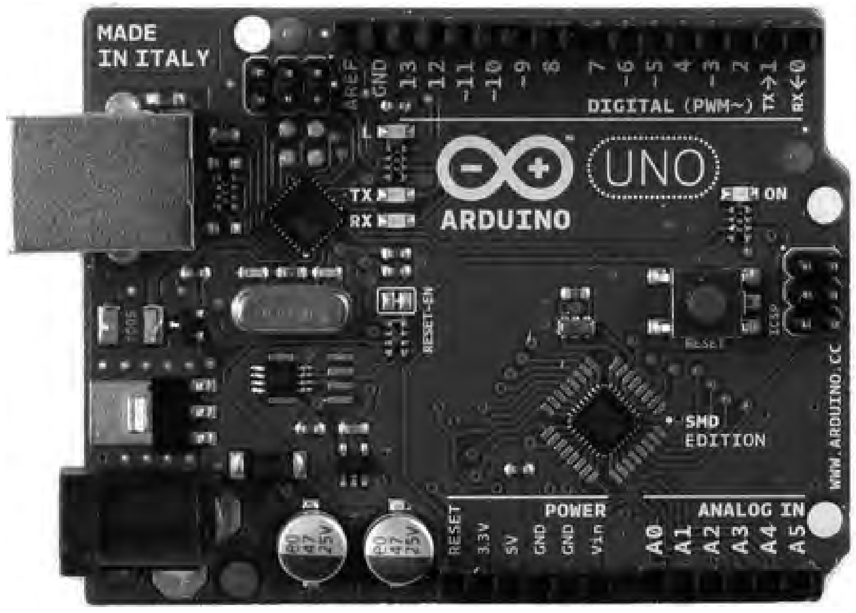


Illustration 1: Arduino Uno

The small board has a USB connector that is used for both programming the device and also as the USB Serial interface.

There are a number of digital inputs, 2 of which can be used to trigger interrupts. The digital inputs can also be programmed to enable internal pull up resistors.

There is also a power connector that is not used as the power is taken from the USB interface.

For this application we want to implement a tuning knob. We can use a step encoder. This uses 2 signal lines to send a clock pulse and a data pulse. These can be used to determine the direction of rotation. I have used a KY-040 encoder. These are available as a small board as can be seen from the image below. They also include a switch that is activated when the button is pressed. These again a readily available. I bought 5 of them for £8 ($12) in the UK.

There are 5 connections, CLK, DATA, SW, +5v and GND. The CLK and DATA lines had 10K pull up resistors on the board. The switch does not but if needed there are pads to add one. The encoder generates 24 pulses per revolution.

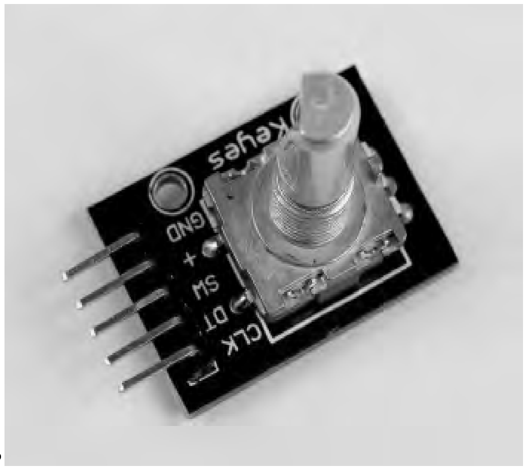


Illustration 2: KY-040 Step Encoder and Switch

Any SPST push to make momentary switches can be used. To make connection simple I soldered a pair of header pins to each switch. For the basic controller I decided on 3 switches.

Illustration 3: Push switch with header pins

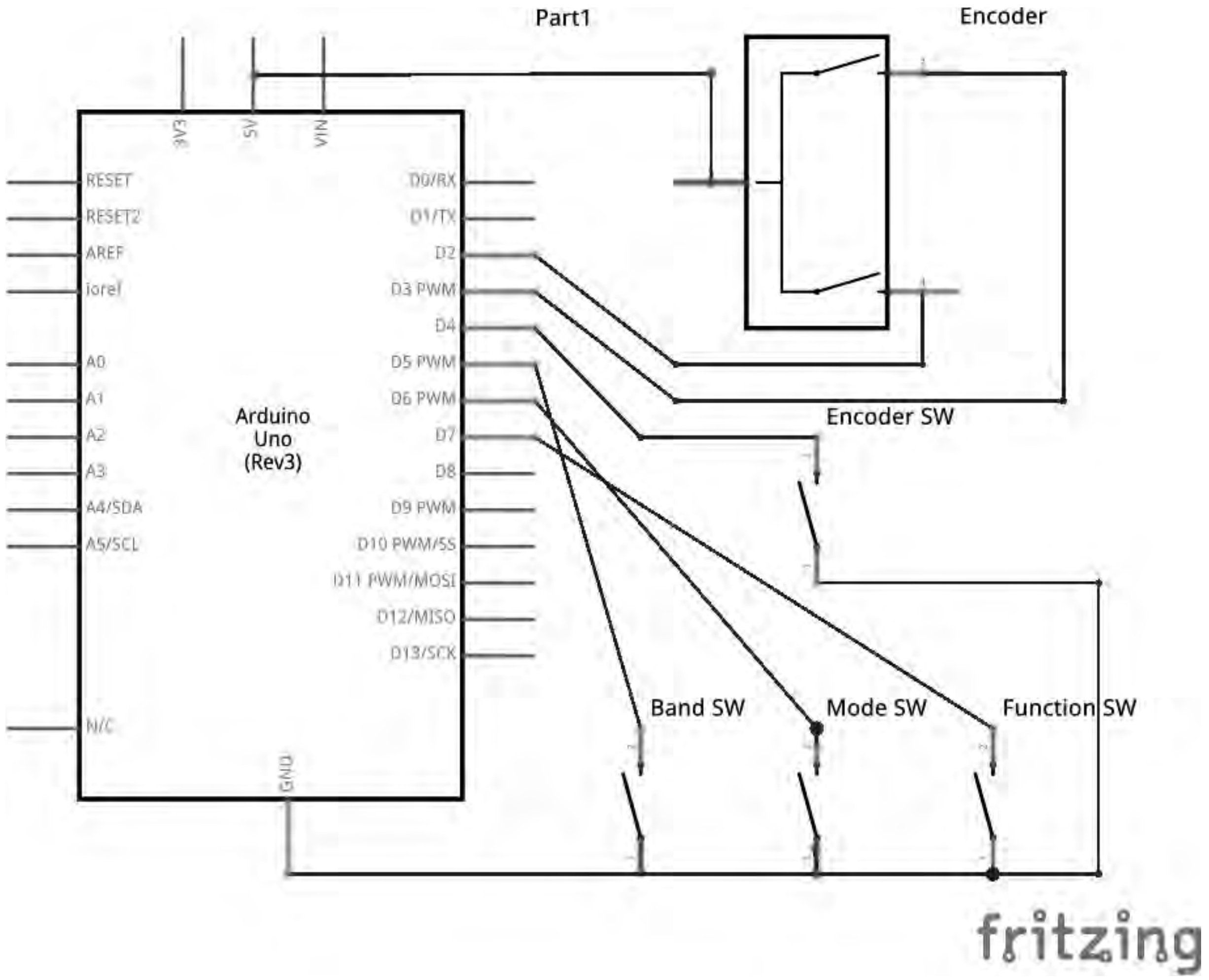
To connect up the components I used some Dupont male to female jumper cables. These usually come as a ribbon cable that can be separated.

Illustration 4: Dupont jumper cables

All that is needed for the hardware is a box to mount the Arduino UNO in with the step encoder and switches mounted on the lid.

Illustration 5: Prototype Controller

Circuit Diagram



The encoders CLK and DATA lines are connected to digital pin D2 and D3 on the Arduino board. Both of these pins can trigger interrupts. This allows fast detection of the steps as it is rotated.

The switch on the encoder is connected to digital pin D4, and the remaining switches are connected to digital pins D5, D6 and D7 on one side and all to GND on the other. All the switches are configured in the software to enable the pull up resistors. This means that the by default they read a a high (1) and go low (0) when pressed.

As you can see there are plenty of pins available to add additional controls.

Software

A *sketch* is the name that Arduino uses for a program. It's the unit of code that is uploaded to and run on an Arduino board. There is an IDE that is used to write the code, compile and upload to the device. A sketch always contains at least 2 functions, setup and loop.

```
void setup() {
}

void loop() {
}
```

The code in the setup function is executed when the device is powered up or reset.

The code in the loop function is then run continually repeating the code.

A simple sketch to configure the USB serial port to run at 9600,N,8,1 and to output the state of a button connected to digital pin 3 every second is shown below:

```
const int buttonPin = 3;

// setup initializes serial and the button pin
void setup()
{
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
}

// loop checks the button pin each time,
// and will send serial if it is pressed
void loop()
{
  if (digitalRead(buttonPin) == HIGH)
    Serial.write('H');
  else
    Serial.write('L');

  delay(1000);
}
```

To save time and effort in development there are libraries available to handle step encoders and debounce switches. I have used Encoder and Bounce2 as they are easily available online and can be loaded into the IDE. These are actually C++ libraries that get compiled along with the sketch.

```
#include <Encoder.h>
#include <Bounce2.h>
```

To define an encoder we simply use the constructor and pass in the pins that the clock and data are connected to. In this case they are pins 2 and 3 so the code in the library will make use of the interrupts available on those lines.

```
Encoder tuningEnc(2, 3);
```

We then need to define the switches using the debounce library. Note that at this time we do not specify the pin as this is done during setup.

```
#define encoderPin 4
Bounce encoderSwitch = Bounce();

#define bandPin 5
Bounce bandSwitch = Bounce();

#define modePin 6
Bounce modeSwitch = Bounce();

#define functionPin 7
Bounce functionSwitch = Bounce();
```

We also need to define some global variables.

```
int encoder=0;  // 1 while encode switch is being pressed
int function=0; // 1 while function switch is being pressed

int afGain = -1;
int rfGain = -1;

char message[8];
int messageIndex=0;
```

The gain values are initially set to -1. This lets the code know it needs to request the current value using CAT commands.

When first powered up or reset the setup function is run. This is used to configure the digital input pins for the switches and the serial port.

```
void setup() {

  // setup function pin
  pinMode(functionPin, INPUT);
  functionSwitch.attach(functionPin);
  functionSwitch.interval(20);
  digitalWrite(functionPin, HIGH);

  // setup band pin
  pinMode(bandPin, INPUT);
```

```
    bandSwitch.attach(bandPin);
    bandSwitch.interval(20);
    digitalWrite(bandPin, HIGH);

    // setup mode pin
    pinMode(modePin, INPUT);
    modeSwitch.attach(modePin);
    modeSwitch.interval(20);
    digitalWrite(modePin, HIGH);

    //setup encoder switch
    pinMode(encoderPin, INPUT);
    encoderSwitch.attach( encoderPin );
    encoderSwitch.interval(20);
    digitalWrite(encoderPin,HIGH);

    Serial.begin(9600);
  }
```

Once the setup function has been run the loop function is run repeatedly. We need to check the status of the switches. One switch is designated as the function switch. When held down it changes the function of the other switches. Note that as the switches are connected to ground and the internal pull up is enabled then they are indicating a high (1) value when not being pressed and a low (0) value when pressed. We process this first before checking the step encoder or other switches as its state determines their functionality. The debounce library has a call to show that the state has changed, Bounce.update, it returns true if it has changed.

```
void loop() {

   …

   if(functionSwitch.update()) {
      if(functionSwitch.read()==0) {
        function=1;
      } else {
        function=0;
      }
    }

    ...

}
```

Another of the buttons is used to change the Band Up/Down. It sends the CAT command to step the band up to the next band if the function switch is not pressed and sends the CAT command to step the band down if it is pressed.

```
if(bandSwitch.update()) {
  if(bandSwitch.read()==0) {
    if(function) {
      Serial.print("ZZBD;");
    } else {
      Serial.print("ZZBU;");
    }
  }
}
```

As you can see, it is fairly simple to implement different CAT commands for different switches.

The tuning control simple sends a CAT command to step the tuning up/down by the current step amount as it receives each step change. The control is also used to change the audio gain if the function button is pressed or the drive level if the step button is pushed in.

One problem is that to change the audio gain or drive level the CAT command contains the value that it is to be changed to. To facilitate this, the first time the audio gain or drive level is changed, the code sees that the current value is -1 so sends a CAT command to retrieve the current value from the host software. Subsequent changes just send the new value. Note that we reset the step position to 0 each time. This means the value returned is always +1 or -1 depending on the direction.

```
long tunePosition = tuningEnc.read();
if (tunePosition != 0) {
  tuningEnc.write(0);
  if(function) {
    if(afGain==-1) {
      Serial.print("ZZAG;"); // get the current audio gain
    } else {
      if(tunePosition<0) {
        afGain--;
        if(afGain<0) {
          afGain=0;
        }
      } else {
        afGain++;
        if(afGain>100) {
          afGain=100;
        }
      }
      Serial.print("ZZAG"); // send the audio gain
      Serial.print(afGain/100);
      Serial.print((afGain%100)/10);
      Serial.print(afGain%10);
      Serial.print(";");
    }
  } else if(encoder) {
    if(rfGain==-1) {
      Serial.print("ZZPC;"); // get the current drive level
    } else {
      if(tunePosition<0) {
        rfGain--;
        if(rfGain<0) {
          rfGain=0;
        }
      } else {
        rfGain++;
        if(rfGain>100) {
          rfGain=100;
        }
      }
      Serial.print("ZZPC"); // send the drive level
      Serial.print(rfGain/100);
      Serial.print((rfGain%100)/10);
      Serial.print(rfGain%10);
      Serial.print(";");
    }
  } else {
    if(tunePosition<0) {
            Serial.print("ZZSB;"); // tuning step back
        } else {
            Serial.print("ZZSA;"); // tuning step forward
        }
      }
   }
```

The final part of the code is to handle serial data received from the host. This is implemented as a function that is called from the loop each time. This reads data 1 byte at a time and builds up the command until it sees a semicolon and then decodes the command.

```
void checkSerialData() {
   while(Serial.available() > 0) {
      // read the incoming byte:
      char c=Serial.read();
      if(c==';') {
        if(strncmp(message,"ZZAG",4)==0 && messageIndex==7) {
            int gain=((message[4]-'0')*100)+
                     ((message[5]-'0')*10)+
                     (message[6]-'0');
            afGain=gain;
        } else if(strncmp(message,"ZZPC",4)==0 && messageIndex==7) {
            int gain=((message[4]-'0')*100)+
                     ((message[5]-'0')*10)+
                     (message[6]-'0');
            rfGain=gain;
        } else if(strncmp(message,"ZZMD",4)==0 && messageIndex==6) {
          int mode=((message[4]-'0')*10)+
                   (message[5]-'0');
          if(function) {
            mode--;
            if(mode<0) {
              mode=11;
            }
          } else {
            mode++;
            if(mode>11) {
              mode=0;
            }
          }
          Serial.print("ZZMD");
          Serial.print(mode/10);
          Serial.print(mode%10);
          Serial.print(";");
        } else {
            // unhandled message;
        }
        messageIndex=0;
      } else {
        message[messageIndex++]=c;
      }
   }
}
```

Conclusion

As I have shown it is fairly simple to implement a CAT based controller for HPSDR. I have shown how to make a fairly limited, but useful, controller at minimal cost. It is not difficult to add additional buttons and step encoders to extend the functionality.

This prototype has 3 step encoders that are configured for audio gain, drive level and tuning. It also has 6 push buttons and a small OLED display. It is also using an Arduino Due for the added performance to support the OLED display.

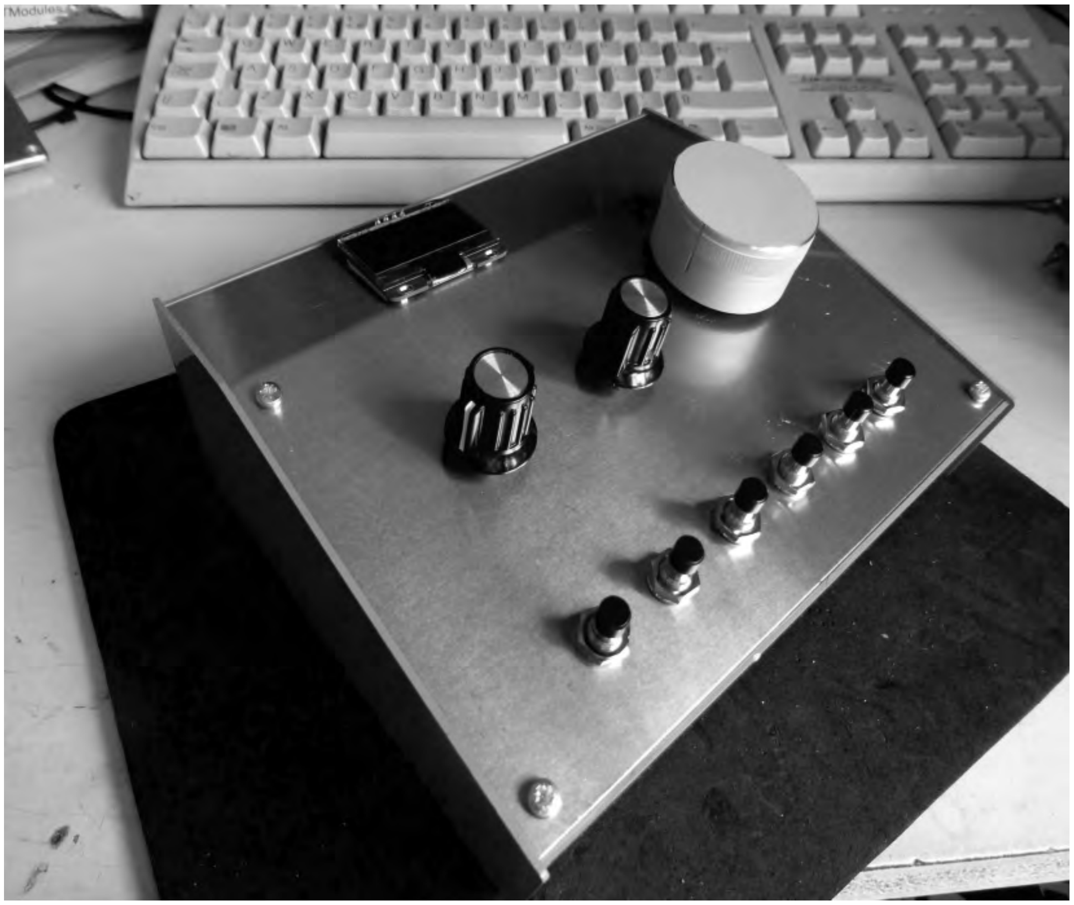


Illustration 6: Prototype controller with more functionality

I have successfully used the controllers with openHPSDR PowerSDR, a modified version of ghpsdr and a modified version of my Android client with some of the CAT commands implemented.

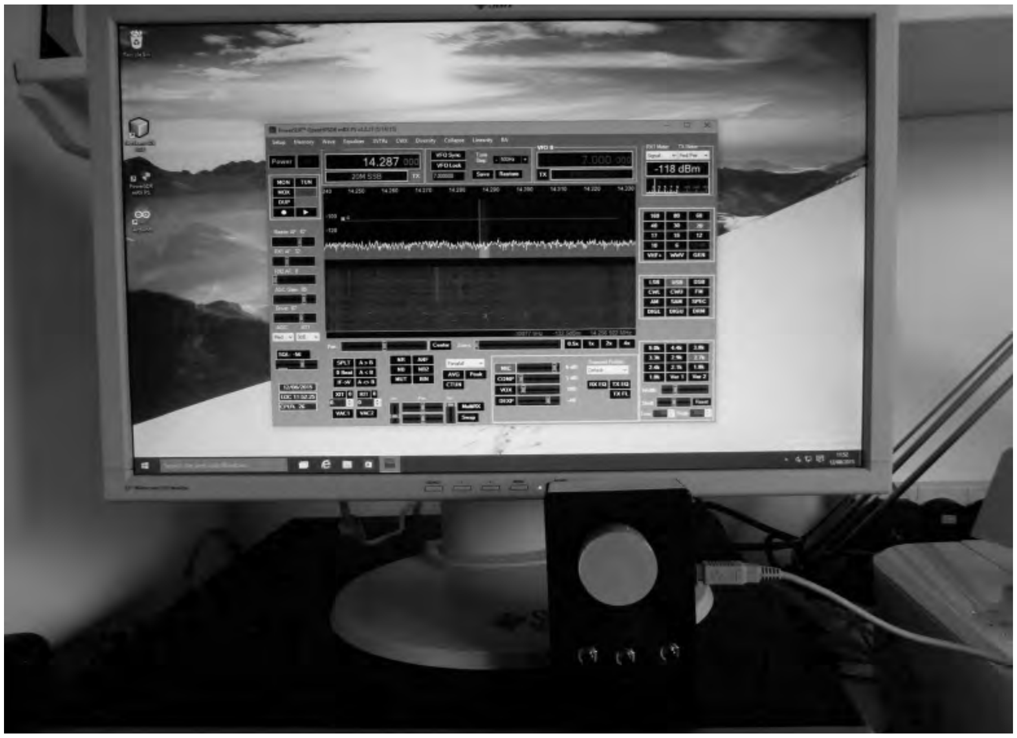


Illustration 7: Windows - openHPSDR PowerSDR

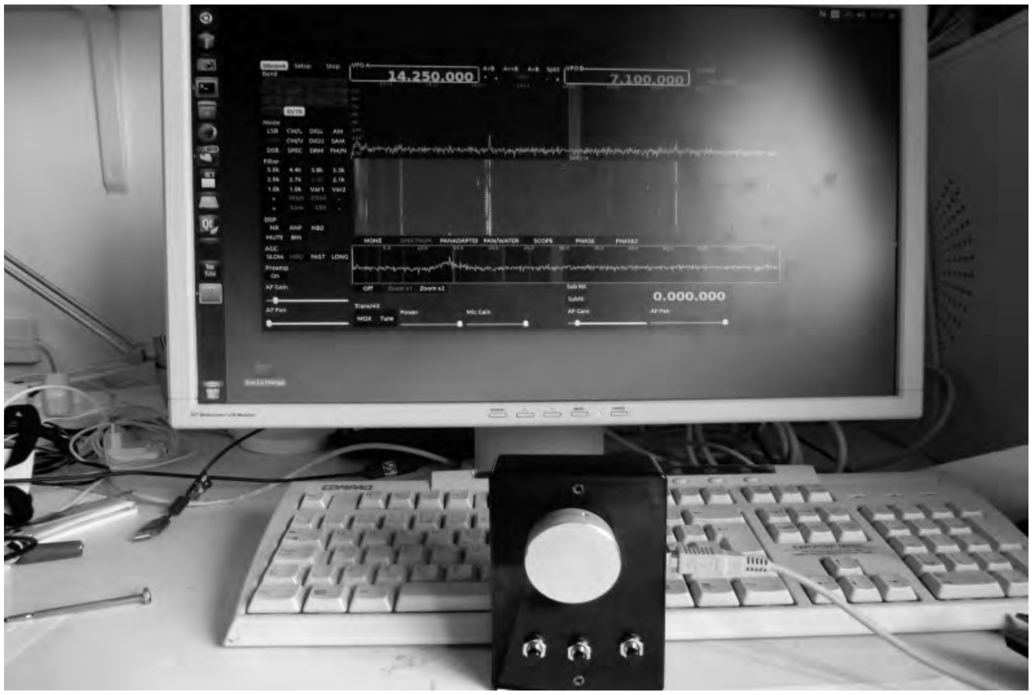


Illustration 8: Linux - ghpsdr

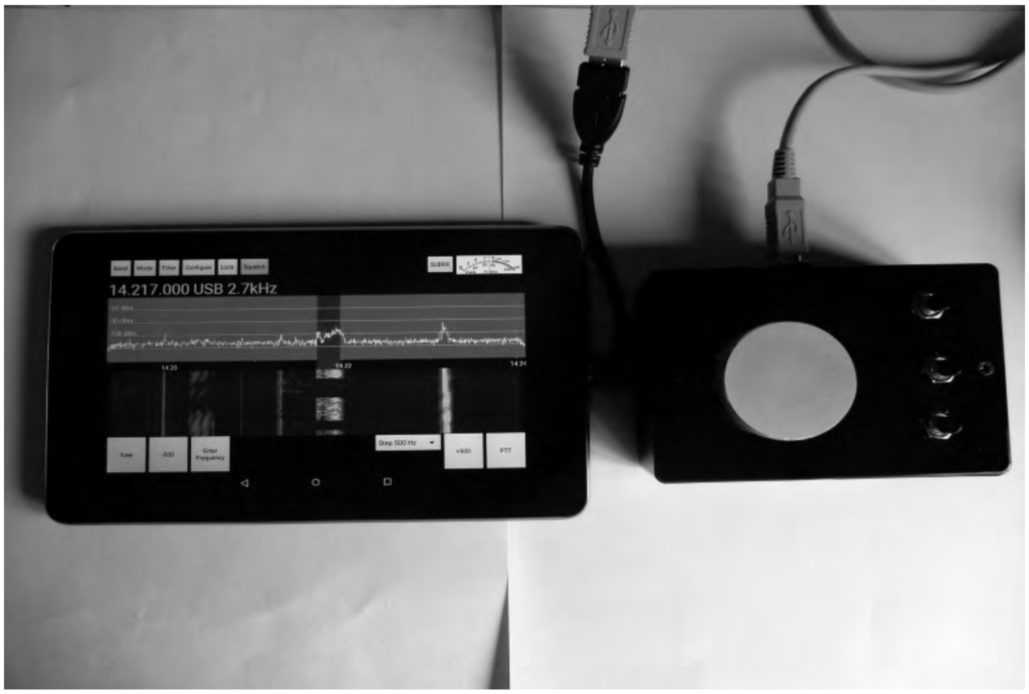


Illustration 9: Android - openHPSDR

References

Arduino: https://www.arduino.cc

Arduino IDE download: https://www.arduino.cc/en/Main/Software

Controller source code: svn.tapr.org/repos_sdr_hpsdr/trunk/N6LYT/Arduino

Encoder library: http://www.pjrc.com/teensy/td_libs_Encoder.html

Switch debounce library: http://playground.arduino.cc/Code/Bounce