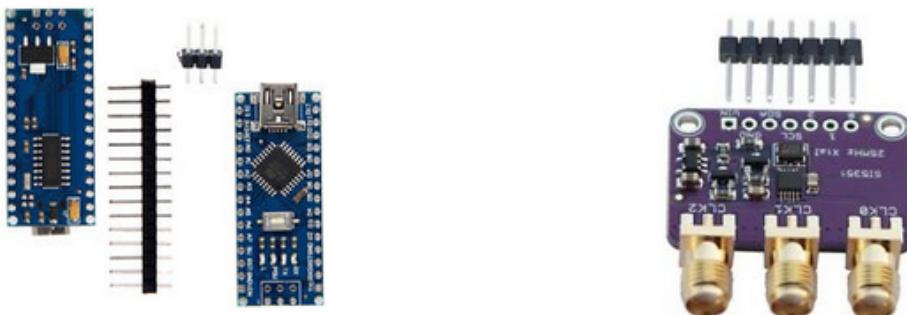


VK3XYZ Arduino VFO Version 1.

This is the first run of an Arduino driven 3 signal capable VFO based around the Arduino nano and the Si5351 VFO board.



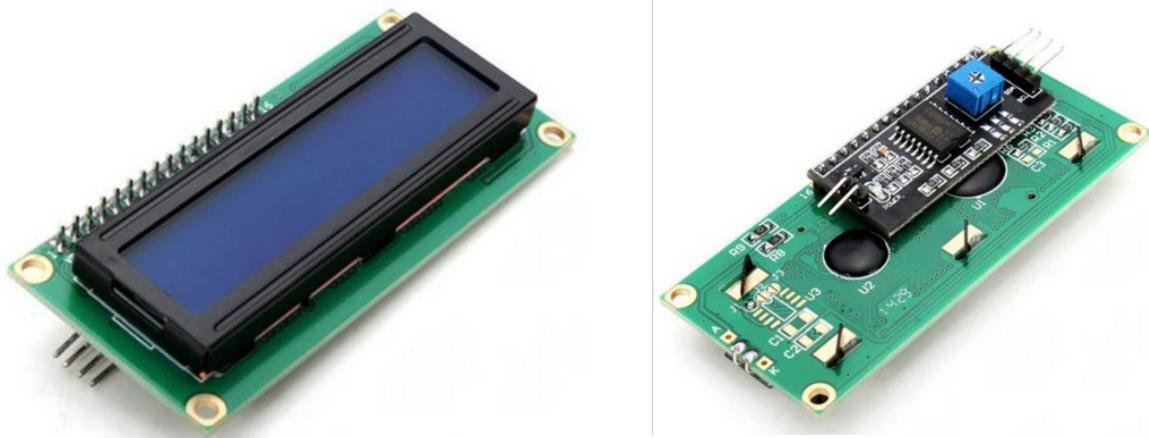
The Arduino Nano is available on Ebay for around \$4.00 and the Si5351 for about \$6.00 so this is a pretty cost effective project.

Tuning is via rotary encoder, one again an Ebay item, available for \$1.00 or less. In fact, the knob will cost you more!



All these add up to a small few dollars.

Then, the display. The board has provision for a parallel 4bit mode LCD and an I2C version. The latter is the one I use.



This can display is a 16 character x 2 lines version but bigger displays can easily be incorporated as required.

A 7805 regulator, polyswitch, tranzorb and a couple of capacitors are the 5V power supply.

2 x optically isolated inputs and 2 optically isolated outputs are for whatever use needed, like PTT input and relay drive output.

The programming is with the free Arduino Integrated Development Environment (IDE) that is freely downloaded from..

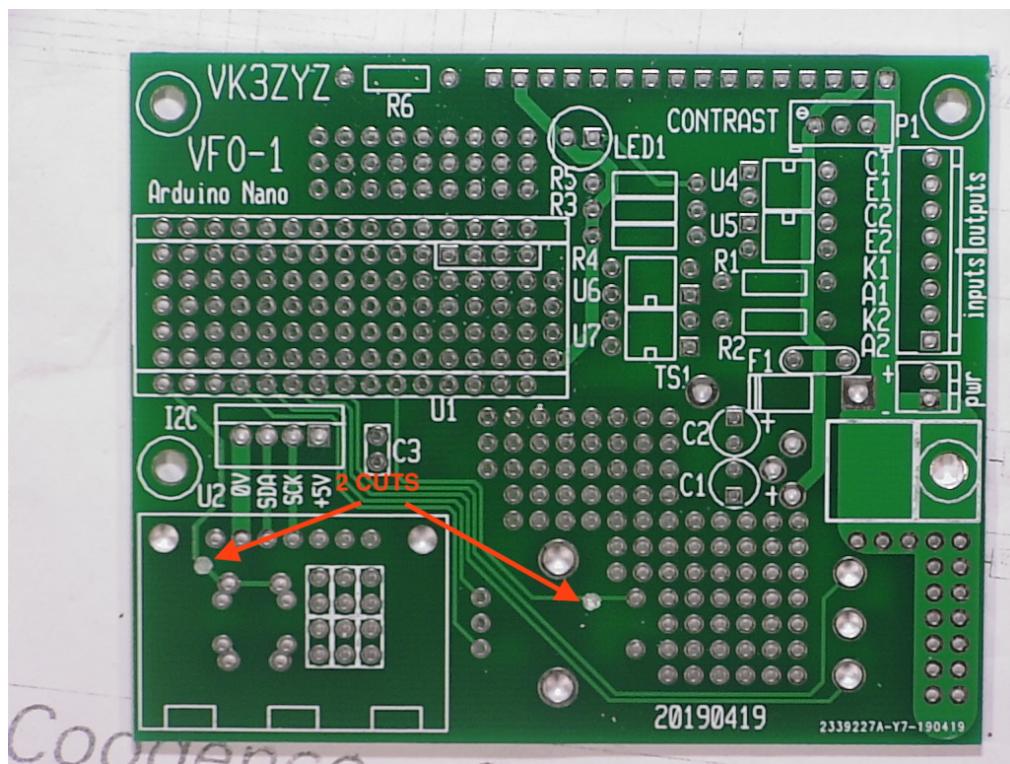
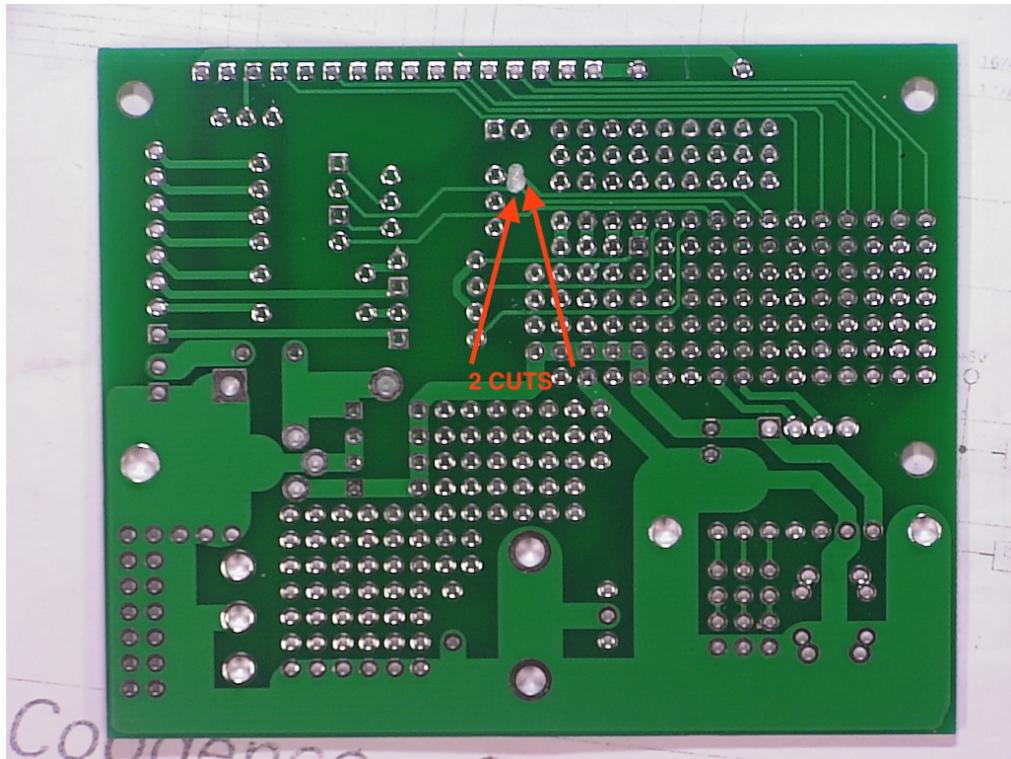
<https://www.arduino.cc/>

A sample program will be provided

VK3XYZ Arduino VFO 20190503

Denys VK3XYZ.

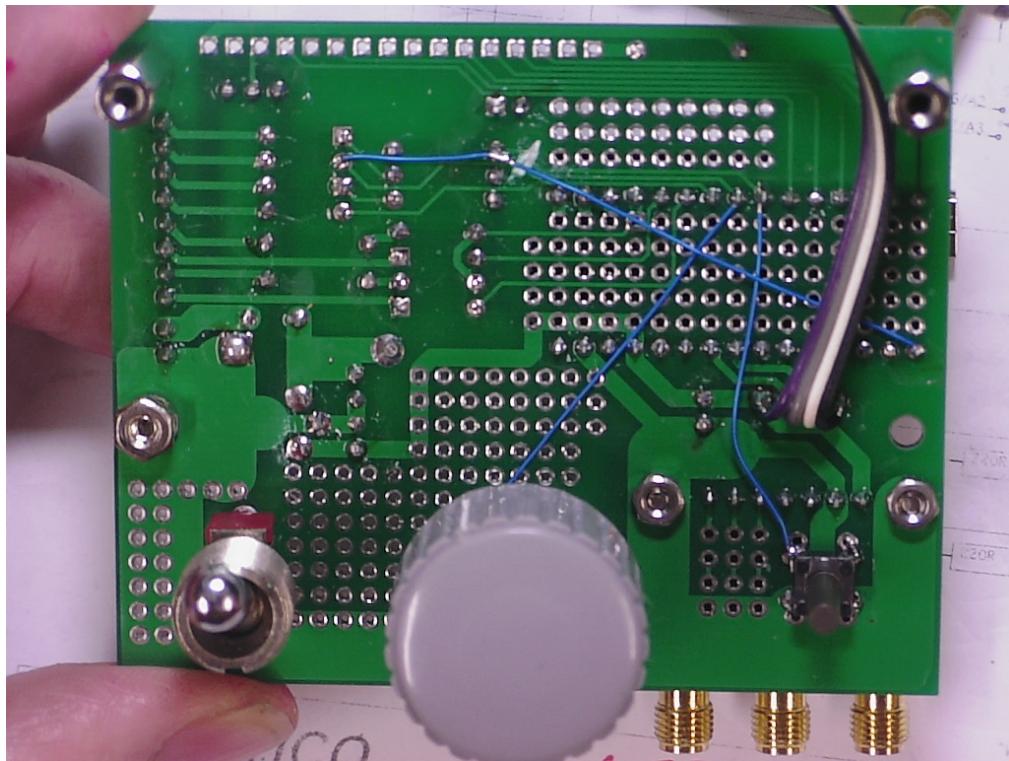
Version1 board needs some mods!
This was a design error, not a layout error.



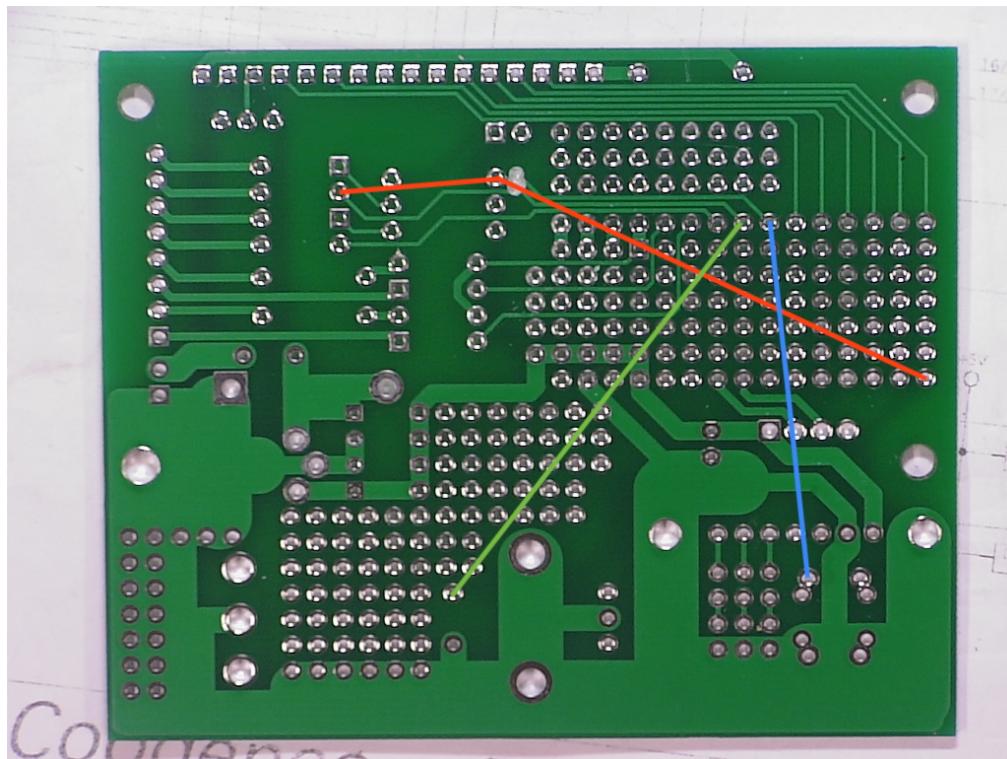
2 cuts on each side.

VK3XYZ Arduino VFO 20190503

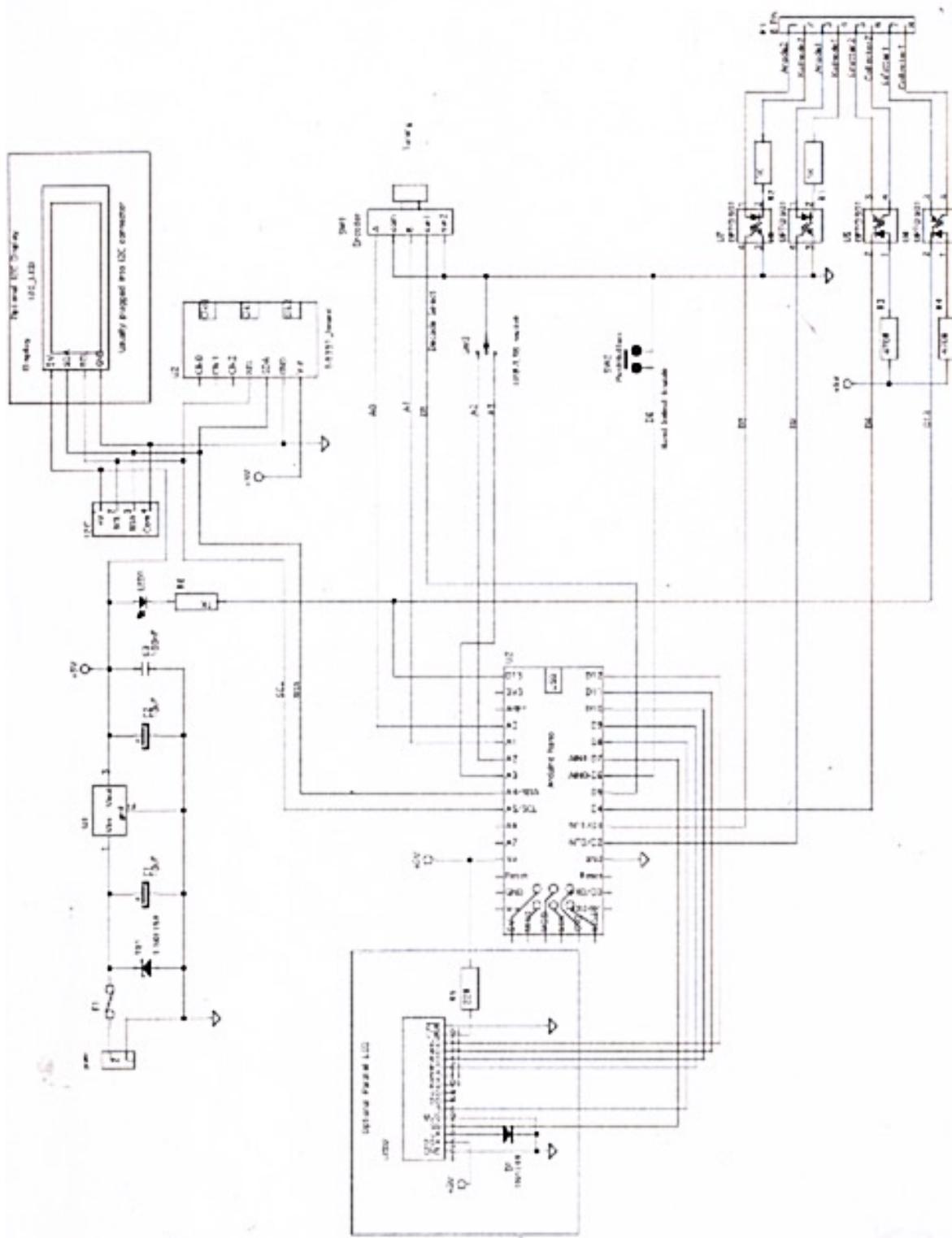
VK3ZYZ Arduino VFO 20190503



Here are the link locations that need to be fitted.



VK3ZYZ Arduino VFO 20190503



Sample code is as follows...

```
/*
Frequ systh for hawk 102-12 radio.
DCP 20190416
Libs as loaded from Etherkit Si5351 examples
Try to add USB/LSB switch
It looks like it works :)
Try to add band limits. :)
next, make 1Hz res.
Add band memory.
*/
#define Version "ArduinoVFO_DCP_5"

#define LCD_I2C // comment out for 4 bits LCD

#include "si5351.h"
#include "Wire.h"

#ifndef LCD_I2C // use this if it is an I2C LCD
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
#else
#include <LiquidCrystal.h> // use this if it is an 4bit LCD
#endif

// synthesiser default settings on start up.
unsigned long frequency = 3630000; // 4 bytes required for frequency

unsigned long frequency0 = 1810000; // 4 bytes required for frequency
unsigned long frequency1 = 3525000; // 4 bytes required for frequency
unsigned long frequency2 = 7000000; // 4 bytes required for frequency
unsigned long frequency3 = 10100000; // 4 bytes required for frequency
unsigned long frequency4 = 14100000; // 4 bytes required for frequency
unsigned long frequency5 = 18068000; // 4 bytes required for frequency
unsigned long frequency6 = 21000000; // 4 bytes required for frequency
unsigned long frequency7 = 24000000; // 4 bytes required for frequency
unsigned long frequency8 = 28000000; // 4 bytes required for frequency

unsigned long step_size = 1000; // default tuning increment (Hz)
unsigned long offset = 1650000; // IF offset for clockgen when driving Kestral/Hawk RFDS Radio
static signed char ham_band_index = 1; // start on 3.5Mhz

long encoder; // encoder direction: CCW=-1; NC=0; CW=1
Si5351 clockgen; // initialize clockgen

// encoder definitions
#define ENCODER_A A0
#define ENCODER_B A1
#define SSB_SWITCH A2 // HIGH = LSB, LOW = USB.
#define AM_SWITCH A3 // not used yet
// A4 = SDA
// A5 = SCL
#define BAND_SENSE A6 // not used yet
#define AnalogIn A7 // not used yet

#define Input1 2 // not used yet
#define Input2 3 // not used yet
#define Output2 4 // not used yet
#define ENCODER_BUTTON 5 // selects tuning decade
#define BAND_SWITCH 6 // When held down, encoder selects band
#define Output1 13 // not used yet
```

VK3ZYZ Arduino VFO 20190503

```
#define ENCODER_INPUT_PORT PINC
int sideband = 0;
int last_sideband = 1;

/* HF Ham bands
Band    UK Allocation      USA Allocation      Australia Allocation
(metres)   Mhz             Mhz             Mhz
160     1.810 - 2.000      1.800 - 2.000      1.800 - 1.875
80      3.500 - 3.800      3.500 - 4.000      3.500 - 3.800
40      7.000 - 7.200      7.000 - 7.300      7.000 - 7.300
30     10.100 - 10.150     10.100 - 10.150     10.100 - 10.150
20     14.100 - 14.350     14.100 - 14.350     14.100 - 14.350
17     18.068 - 18.168     18.068 - 18.168     18.068 - 18.168
15     21.000 - 21.450     21.000 - 21.450     21.000 - 21.450
12     24.890 - 24.990     24.890 - 24.990     24.890 - 24.990
10     28.000 - 29.700     28.000 - 29.700     28.000 - 29.700
*/
#endif LCD_I2C // use this if it is an I2C LCD

//LiquidCrystal_I2C lcd(0x20,4,5,6,0,1,2,3); // <<<<< Works with the GY-IICLCD board. "bl" and "bipol" not used.
//LiquidCrystal_I2C lcd(0x3f,2,1,0,4,5,6,7,3,POSITIVE); // 0x3f is the I2C bus address for PCF8574AT HM Board
LiquidCrystal_I2C lcd(0x27,2,1,0,4,5,6,7,3,POSITIVE); // 0x27 is the I2C bus address for PCF8574T HM Board

#else // use this if it is an 4bit LCD
/*
The circuit:
* LCD RS pin to digital pin 7
* LCD Enable pin to digital pin 8
* LCD D4 pin to digital pin 9
* LCD D5 pin to digital pin 10
* LCD D6 pin to digital pin 11
* LCD D7 pin to digital pin 12
* LCD R/W pin to ground
* LCD VSS pin to ground
* LCD VCC pin to 5V
* initialize the library with the numbers of the interface pins
*/
//      RS, E, D4, D5, D6, D7
LiquidCrystal lcd(7, 8, 9, 10, 11, 22);
#endif

void setup() {

lcd.begin(16,2);          // 16 char x 2 line LCD display
lcd.print(Version);
delay(1000);
lcd.clear();

bool i2c_found;

// Start serial and initialize the Si5351
Serial.begin(57600);
i2c_found = clockgen.init(SI5351_CRYSTAL_LOAD_8PF, 0, 0);
if(!i2c_found)
{
  lcd.print("Device not found on I2C bus!");
}

// configure rotary encoder & push-button
pinMode(ENCODER_A, INPUT_PULLUP); // encoder leads
pinMode(ENCODER_B, INPUT_PULLUP);
pinMode(ENCODER_BUTTON, INPUT_PULLUP);
pinMode(BAND_SWITCH, INPUT_PULLUP);
```

VK3ZYZ Arduino VFO 20190503

```
pinMode(SSB_SWITCH, INPUT_PULLUP); // Hi = LSB, LOW = USB
pinMode(AM_SWITCH, INPUT_PULLUP);
pinMode(Input1, INPUT_PULLUP);
pinMode(Input2, INPUT_PULLUP);
pinMode(Output1,OUTPUT);
pinMode(Output2,OUTPUT);
digitalWrite(Output1,HIGH); // output1 off
digitalWrite(Output2,HIGH); // output2 off

display_freq_step(); // display frequency & step-size
set_synth();
}

/******************
main loop
*****************/
void loop() {
check_band();
check_step_size();

if (digitalRead(SSB_SWITCH)!=last_sideband)
{ set_synth();
  display_freq_step(); // display frequency & step-size
}
// check the frequency
encoder = read_encoder(); // returns +1/0/-1 (cw/nc/ccw)
if (encoder != 0) {
  frequency = frequency + (encoder * step_size);

  set_synth();
  display_freq_step(); // display frequency & step-size
}

/******************
read encoder direction
*****************/
char read_encoder() {
// definitions
#define DELAY 100 // number of polling-loops
static unsigned short integrator; // encoder debounce

/* the array values are determined by combining the previous and current A1,A0
Gray Code bit-patterns to form an array-index. +1 is assigned to all clockwise
(CW) indexes; -1 is assigned to all counter-clockwise (CCW) indexes; and 0 is
assigned to the remaining positions to indicate no movement*/
static char encoder[] = {0,-1,1,0,1,0,0,-1,-1,0,0,1,0,1,-1,0}; // SWAPPED PHASES
// static char encoder[] = {0,1,-1,0,-1,0,0,1,1,0,0,-1,0,-1,1,0};
static unsigned char encoder_index = 0;

static long count = 0; // CW rotation = +1; CCW rotation = -1
static long this_reading;
static long last_reading;
char value; // return value

/* check for encoder rotation. The absolute "count" value changes as the encoder rotates.
Typically there are four states (counts) per click, but often more due to contact bounce.
Bounce doesn't matter so long as the "count" decreases" for CCW rotation; remains constant
when there is no rotation; and increases with CW rotation. Bounce can be completely
eliminated if we look for the "detent" code and integrate as shown below.*/
encoder_index <= 2; // shift previous bit-pattern left two places
encoder_index |= (ENCODER_INPUT_PORT & 0x03); // get current bit-pattern and combine with previous
count += encoder[(encoder_index & 0x0f)];
```

VK3XYZ Arduino VFO 20190503

```
// debounce encoder
if((ENCODER_INPUT_PORT & 0x03) == 3) {      // encoder on an indent (3 decimal == 00000011 binary)
    if(integrator < DELAY) {
        integrator++;
    }
} else {                                // encoder not resting on indent
    if(integrator > 0) {
        integrator--;
    }
}

// prepare the "return" value
if(integrator >= DELAY) {                // encoder deemed to be stationary
    this_reading = count;
    if(this_reading < last_reading) value = -1;
    if(this_reading == last_reading) value = 0;
    if(this_reading > last_reading) value = 1;
    last_reading = this_reading;           // both readings now hold current "count"
    integrator = 0;                      // reset integrator
} else {
    value = 0;
}
return(value);
}

/******************
check frequency band
*****************/
void check_band() {
#define DELAY 100                  // number of polling-loops
static unsigned short integrator;        // encoder debounce

static char encoder[] = {0,-1,1,0,1,0,0,-1,-1,0,0,1,0,1,-1,0}; // SWAPPED PHASES
// static char encoder[] = {0,1,-1,0,-1,0,0,1,1,0,0,-1,0,-1,1,0};
static unsigned char encoder_index = 0;
static long count = 0;                  // CW rotation = +1; CCW rotation = -1
static long this_reading;
static long last_reading;

long ham_band[] = {frequency0,frequency1,frequency2,frequency3,frequency4,frequency5,frequency6,frequency7,frequency8};

while (digitalRead(BAND_SWITCH) == LOW) {
    // check for encoder rotation
    encoder_index <= 2;
    encoder_index |= (ENCODER_INPUT_PORT & 0x03);
    count += encoder[(encoder_index & 0x0f)];

    if((ENCODER_INPUT_PORT & 0x03) == 3) { // encoder on indent = B00000011
        if(integrator < DELAY) {
            integrator++;
        }
    } else {                                // encoder not resting on indent
        if(integrator > 0) {
            integrator--;
        }
    }

    if(integrator >= DELAY) {                // encoder stationary
        this_reading = count;
        if(this_reading != last_reading) {
            if(this_reading > last_reading) {
                ham_band_index++;
                if(ham_band_index > 8) {
                    ham_band_index = 0;          // wrap around
                }
            }
        }
    }
}
```

VK3ZYZ Arduino VFO 20190503

```
        }
    }
    if (this_reading < last_reading) {
        ham_band_index--;
        if (ham_band_index < 0) {
            ham_band_index = 8;
        }
    }
    last_reading = this_reading;           // both now hold current count
    integrator = 0;                      // reset integrator
    frequency = ham_band[ham_band_index]; // change frequency band
    set_synth();                         // set the new freq ***** added 20190502
    display_freq_step();
}
}
}

/******************
check the step size
*****************/
void check_step_size() {
    // delay definitions
#define DELAY 3000                  // DELAY = debounce(mS)*loops/second
#define LONG_PUSH 15000             // number of loops considered "long"

    // variables
    static boolean transition = false; // valid during button push
    static unsigned short integrator; // varies from 0 to MAXIMUM
    static unsigned short push_duration; // determines step size/direction

    // look for first high-to-low transition
    if (!transition && (digitalRead(ENCODER_BUTTON) == LOW)) {
        transition = true;
        integrator = 0;
        push_duration = 0;
    }

    // look for last low-to-high transition
    if (transition) {
        if (push_duration < (LONG_PUSH*2)) { // track push duration
            push_duration++;
        }
        if (digitalRead(ENCODER_BUTTON) == HIGH) { // ENCODER_BUTTON is HIGH
            if (integrator < DELAY) {
                integrator++;
            } else { // ENCODER_BUTTON is LOW
                if (integrator > 0) {
                    integrator--;
                }
            }
        }
    }

    // update the step_size
    if (integrator >= DELAY) {
        transition = false;
        if (push_duration > LONG_PUSH) { // LONG PUSH: decrease step size
            step_size = step_size/10;
            if (step_size < 1) { // was 10
                step_size = 100000; // wrap around
            }
        } else { // SHORT PUSH: increase step size
            step_size = step_size*10;
        }
    }
}
```

VK3ZYZ Arduino VFO 20190503

```
if (step_size > 100000) {
    step_size = 1;           // wrap around Was 10
}
}

display_freq_step();      // update step-size
}

}

*****  
set the Si5351 CLK0 frequency  
*****  
void set_synth ()  
{  
switch (ham_band_index) {    // limit to SSB part of bands  
    case 0:  
frequency = constrain(frequency, 1840000, 1875000);  
frequency0 = frequency;  
break;  
  
    case 1:  
frequency = constrain(frequency, 3525000, 3700000);  
frequency1 = frequency;  
break;  
  
    case 2:  
frequency = constrain(frequency, 7040000, 7300000);  
frequency2 = frequency;  
break;  
  
    case 3:  
frequency = constrain(frequency, 10115000, 10140000);  
frequency3 = frequency;  
break;  
  
    case 4:  
frequency = constrain(frequency, 14112000, 14350000);  
frequency4 = frequency;  
break;  
  
    case 5:  
frequency = constrain(frequency, 18110000, 18168000);  
frequency5 = frequency;  
break;  
  
    case 6:  
frequency = constrain(frequency, 21150000, 21450000);  
frequency6 = frequency;  
break;  
  
    case 7:  
frequency = constrain(frequency, 24930000, 24990000);  
frequency7 = frequency;  
break;  
}  
  
if (digitalRead(SSB_SWITCH)==LOW)
```

VK3ZYZ Arduino VFO 20190503

```
{  
clockgen.set_freq(((frequency+offset)*100), SI5351_CLK0);  
last_sideband=0;  
}  
else  
{  
clockgen.set_freq(((frequency-offset)*100), SI5351_CLK0);  
last_sideband=1;  
}  
  
*****  
update frequency and step-size  
*****  
void display_freq_step()  
long MHz = frequency/1000000;  
  
// convert frequency to floating point  
// adding 0.0000001 fixes a rounding error  
double freq = (double) frequency/1000000+0.0000001;  
  
// display frequency  
lcd.clear();  
if (digitalRead(SSB_SWITCH)==HIGH)  
{ lcd.print("L");  
}  
else  
{ lcd.print("U");  
}  
  
if (MHz < 10) {  
lcd.setCursor(3,0); // start at char 1, line 1  
lcd.print(freq, 6); // 6 digit resolution  
lcd.print(" MHz ");  
} else {  
lcd.setCursor(2,0); // start at char 1, line 1  
lcd.print(freq, 6);  
lcd.print(" MHz ");  
}  
  
// commented out to set 1x Hz.  
// last digit always zero // digit sometimes showed a 1  
// lcd.setCursor(10,0);  
// lcd.print("0");  
  
// display step size  
switch (step_size) {  
  
case 1: // added for 1xHz.  
lcd.setCursor(10,1);  
lcd.print("*");  
break;  
  
case 10:  
lcd.setCursor(9,1);  
lcd.print("*");  
break;  
case 100:  
lcd.setCursor(8,1);  
lcd.print("*");  
break;  
case 1000:  
lcd.setCursor(7,1);  
}
```

VK3XYZ Arduino VFO 20190503

```
lcd.print("*");
break;
case 10000:
lcd.setCursor(6,1);
lcd.print("*");
break;
case 100000:
lcd.setCursor(5,1);
lcd.print("*");
break;
default:
break;
}
}
```