```
//SmithChart Tuner SPI Graphic 03/25/2020

/***************************************************
  This is our library for the Adafruit HX8357D Breakout
  ----> http://www.adafruit.com/products/2050

  Check out the links above for our tutorials and wiring diagrams
  These displays use SPI to communicate, 4 or 5 pins are required to
  interface (RST is optional)
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  MIT license, all text above must be included in any redistribution
 ****************************************************/

#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SPITFT.h>
#include <Adafruit_SPITFT_Macros.h>
#include <gfxfont.h>
#include <stdint.h>
#include "TouchScreen.h"
#include "Adafruit_GFX.h"
#include "Adafruit_HX8357.h"

#define YP A5  // must be an analog pin, use "An" notation!
#define XM A4  // must be an analog pin, use "An" notation!
#define YM 2   // can be a digital pin
#define XP 3   // can be a digital pin

// These are 'flexible' lines that can be changed
#define TFT_CS 10
#define TFT_DC 9
#define TFT_RST 8 // RST can be set to -1 if you tie it to Arduino's reset

// Use hardware SPI (on Uno, #13, #12, #11) and the above for CS/DC
TouchScreen ts = TouchScreen(XP, YP, XM, YM, 300);
Adafruit_HX8357 tft = Adafruit_HX8357(TFT_CS, TFT_DC, TFT_RST);

// SoftSPI - note that on some processors this might be *faster* than hardware SPI!
//Adafruit_HX8357 tft = Adafruit_HX8357(TFT_CS, TFT_DC, MOSI, SCK, TFT_RST, MISO);
#define BLACK 0x0000
#define BLUE 0x001F
#define RED 0xF800
#define GREEN 0x07E0
#define CYAN 0x07FF
#define MAGENTA 0xF81F
#define YELLOW 0xFFE0
#define WHITE 0xFFFF

#define PM  A0 //Voltage magnitude wrt Current magnitude in dB
#define VM A1 //Phase in degrees
#define PMB A2  //phase shift signal
#define Pin80m 7  //80 meter phase shifter relays
#define Pin40m 6  //40 meter phase shifter relays
#define Pin20m 5  //20 meter phase shifter relays

float Phase;  //Phase between to signals +/- 180 degrees
float VMMagVal = 0;
float ZReal;  //Real part of impedance
float ZIm;  //Imaginary part of impedance
float GammaMag; //Reflection coefficient magnitude (0 to 1)
float GammaAngle; //Reflection coefficient angle (0 to 180 and -180 to 0)
unsigned long CurrentTime;  //Time in milliseconds since program start
unsigned long PreviousTime; //Previous time from 5 second mark
int PMMag;
int PMMagB;
int PMMagBPrev;
int PhaseFlag = 0;
int Xcordinate;
int Ycordinate;
int Relay80 = 7;  //80m 90 degree Phase shifter control
int Relay40 = 6;  //40m 90 degree Phase shifter control
int Relay20 = 5;  //20m 90 degree Phase shifter control
int Band;

void setup() {
  // put your setup code here, to run once:
Serial.begin(9600);
//analogReference;
```

```
Serial.println("Smith Chart Tune");

tft.begin();
tft.setRotation(1);
Logo();
delay(2000);
tft.fillScreen(WHITE);
DrawSmithChart();
PreviousTime = millis();
PMMagBPrev = PMMagB;
Label_20m_Red();
Label_40m_Red();
Label_80m_Red();

pinMode(Relay80, OUTPUT); //80m relay control
pinMode(Relay40, OUTPUT); //40m relay control
pinMode(Relay20, OUTPUT); //20m relay control
digitalWrite(Relay80, LOW); //Open 80m 90 degree phase shift circuit
digitalWrite(Relay40, LOW); //Open 40m 90 degree phase shift circuit
digitalWrite(Relay20, LOW); //Open 40m 90 degree phase shift circuit

}

void loop()
{
  analogReference(EXTERNAL);
  Plot();
  CurrentTime = millis(); //Get current time in milliseconds
  //Update every SWR, Impedance, Smith Chart, and Reflection Coeffcient every 5 seconds
  if(CurrentTime - PreviousTime >= 5000)
  {
    tft.fillRect(0, 300, 480, 20, WHITE); //Erase previous SWR and Impedance
    SWRCalc();  //Calculate and print SWR
    ZPrint();  //Prints impedance
    DrawSmithChart(); //Redraw Smith Chart
    PreviousTime = CurrentTime; //Save 5 second time mark
  }
  analogReference;
  TouchScreenSelect();
}

//Logo displays the W0IVJ Logo
void Logo()
{
  tft.fillScreen(BLUE); //Fill screen with blue
  TextParameters(5, YELLOW, BLUE);  //Set the text parameters for Yellow with blue background
  CharacterCursorPosition(5, 1, 5);
  tft.print("W0IVJ");
  CharacterCursorPosition(5, 3, 2);
  tft.print("SMITH CHART");
  CharacterCursorPosition(5, 5, 5);
  tft.print("TUNER");
}

// Normalizes, Calculates, and Plots the Smith Chart point
void Plot()
{
  int VMMag = 0;
  float VMMagAdj = 0;
  float PMMagVal;
  float PMMagBVal;
  float Vref = 1.83;
  float Vref25 = 0.4575;
  float Vref5 = 0.915;
  float Vref75 = 1.3725;
  float XVal;
  float YVal;
  int XValPlot = 240;
  int YValPlot = 160;
  float R;
  float X;
  int n = 0;
  int y;
  int PhaseSign;
  float VCum = 0;
  float PCum = 0;
  float PCumB = 0;
  float ZMag;
  float VdB;
  float Gamma_N_Re; //Real part of Gamma numerator
  float Gamma_N_Im; //Imaginary part of Gamma numerator
  float Gamma_N_P_Mag;  //
```

```
 float Gamma_N_P_Angle;
 float Gamma_D_Re; //Real part of Gamma
 float Gamma_D_Im; //Imaginary part of Gamma
 float Gamma_D_P_Mag;  //Gamma magnitude
 float Gamma_D_P_Angle;  //Gamma angle in degrees

 for(n = 0; n < 10; n++)
 {
   VMMag = analogRead(VM); //Get magnitude
   VCum = VCum + VMMag;  //Accumulate magnitude
 }
 VMMag = VCum / 10;  //Average magnitude
 for(n = 0; n < 10; n++)
 {
   PMMag = analogRead(PM); //Get pahse
   PCum = PCum + PMMag;  //Accumulate phase
 }
 PMMag = PCum / 10;  //Average phase
 PMMagVal = PMMag / 511.5;

for(n = 0; n < 10; n++)
 {
   PMMagB = analogRead(PMB); //Get pahse shift
   PCumB = PCumB + PMMagB;  //Accumulate phase shift
 }
 PMMagB = PCumB / 10;  //Average phase shift
 PMMagBVal = PMMagB / 511.5;



 if((PMMagVal >= Vref75) && (PMMagBVal <= Vref5))
 {
   Phase = ((PMMagBVal - 0.9) / -.01);
 }
 else if(PMMagBVal < Vref25)
 {
   Phase = ((PMMagVal - 0.9) / -.01) + 90;
 }
 else if(PMMagVal <= Vref25)
 {
   Phase = ((PMMagBVal - 0.9) / .01) + 180;
 }
 else if(PMMagBVal > Vref75)
 {
   Phase = ((PMMagVal - 0.9) / .01) +270;
 }
 else if((PMMagVal >= Vref75) && (PMMagBVal >= Vref5))
 {
   Phase = ((PMMagBVal - 0.9) / -0.01) +360;
 }

 if(Band == 80)
   {
     int PhaseCalc = 0.9889 * Phase +1.0011;
     Phase = PhaseCalc;
     if(Phase >= 180)
     {
       Phase = 0.1863 * PMMag +179.18;
       Phase = Phase + 5;
     }
     else if(Phase < 180)
     {
       Phase = -0.1871 * PMMag + 178.32;
       Phase = Phase - 2;
     }
   if((VMMag >= 224) && (VMMag <= 766))  //Cal for -15 dB to +20 dB
   {
     VdB = 0.0641 * VMMag - 29.043;
   }
   else if(VMMag > 766)  //Cal for +20 dB to +30 dB
   {
     VdB = .0627 * VMMag - 28.049;
   }
   else if (VMMag < 224) //Cal for -15 dB to -30 dB
   {
     VdB = -0.0012 * VMMag * VMMag + 0.5538 * VMMag - 78.687;
   }
 VdB = -(VdB + 1); //This compensates for loss and which input was attenuated during calibration
 }

 else if(Band == 40)
 {
```

```
      int PhaseCalc = 0.988 * Phase + 0.557;
      Phase = PhaseCalc;
      if(Phase >= 180)
      {
        Phase = 0.1871 * PMMag +177.27;
      Phase = Phase + 6;  //Final tweak around zero
      }
      else if(Phase < 180)
      {
        Phase = -0.1883 * PMMag + 177.28;
      Phase = Phase - 2;  //Final tweak around zero
      }


     if((VMMag >= 231) && (VMMag <= 775)) //Cal for -15 dB to +20 dB
    {
      VdB = 0.0639 * VMMag - 29.385;
    }
    else if(VMMag > 775)  //Cal for +20 dB to +30 dB
    {
      VdB = 0.0614 * VMMag - 27.594;
    }
    else if (VMMag < 231) //Cal for -15 dB to -30 dB
    {
      VdB = -0.0017 * VMMag * VMMag + 0.7682 * VMMag - 101.95;
    }
   VdB = -(VdB + 1.3); //This compensates for loss and which input was attenuated during calibration
  }
  else if(Band == 20)
  {
      int PhaseCalc = 0.9869 * Phase -0.5964;
      Phase = PhaseCalc;
      if(Phase >= 180)
      {
        Phase = 0.1897 * PMMag +173.51;
       Phase = Phase + 10;  //Final tweak around zero
      }
      else if(Phase < 180)
      {
        Phase = -0.1914 * PMMag + 175.99;
      }

    if((VMMag >= 238) && (VMMag <= 781))  //Cal for -15 dB to +20 dB
    {
      VdB = 0.064 * VMMag - 29.624;
    }
    else if(VMMag > 781)  //Cal for +20 dB to +30 dB
    {
      VdB = 0.0537 * VMMag -21.743;
    }
    else if (VMMag < 238) //Cal for -15 dB to -30 dB
    {
      VdB = -0.0019 * VMMag * VMMag +0.8774 * VMMag -119.18;
    }
   VdB = -(VdB + 1); //This compensates for loss and which input was attenuated during calibration
  }

if(Phase > 180)
{
  Phase = Phase - 360;
}
ZMag = (50 * pow(10, (VdB / 20)));  //Calculate the real part of impedance
ZReal = abs(ZMag * cos(Phase / 57.2958)); //Convert real part of Z from polar to rectangular cooridinates
ZIm = ZMag * sin(Phase / 57.2958); //Convert imaginary part of Z from polar to rectangular cooridinates

//Calculate Reflection Coefficient(Gamma) from Impedance
Gamma_N_Re = (ZReal - 50);  //Calculate the numerator of the real part of Gamma
Gamma_N_Im = ZIm; //Calculate the numerator of the imaginary part of Gamma
Gamma_N_P_Mag = sqrt(Gamma_N_Re * Gamma_N_Re + Gamma_N_Im * Gamma_N_Im);
Gamma_N_P_Angle = atan(Gamma_N_Re / Gamma_N_Im) * 57.2958;

Gamma_D_Re = ZReal + 50;  //Calculate real part of Gamma
Gamma_D_Im = ZIm; //Calculate imaginary part of Gamma
Gamma_D_P_Mag = sqrt(Gamma_D_Re * Gamma_D_Re + Gamma_D_Im * Gamma_D_Im);
Gamma_D_P_Angle = atan(Gamma_D_Re / Gamma_D_Im) * 57.2958;

GammaMag = Gamma_N_P_Mag / Gamma_D_P_Mag; //Calculate the magnitude of Gamma
if(GammaMag > 1) {GammaMag = 1;}  //Gamma cannot be greater than 1
GammaAngle = Gamma_N_P_Angle - Gamma_D_P_Angle; //Calculate the angle of Gamma in degrees

//Convert Gamma from polar to rectangular coordinates
XVal = GammaMag * cos(GammaAngle / 57.2958);
```

```
YVal = GammaMag * sin(GammaAngle / 57.2958);

//Convert X to pixels
XValPlot = (XVal * 160) + 240; //Convert to pixels
//Convert Y to pixels
YValPlot = (YVal * 160) + 160; //Convert to pixels



for(n = 0; n < 200; n++)
{
tft.drawRect(XValPlot-2, YValPlot-3, 6, 6, BLACK);  //Plot new Gamma value (3.1 ms total time)
}
tft.drawRect(XValPlot-2, YValPlot-3, 6, 6, WHITE);  //Erase Gamma value
}

//sets text size, text color and background color
void TextParameters(uint8_t size, uint16_t color, uint16_t backgroundcolor)
{
  tft.setTextColor(color, backgroundcolor);
  tft.setTextSize(size);
}

//Positions cursor with respect to size, line number and letter position
void CharacterCursorPosition(uint8_t size, uint16_t LineNumber, uint16_t LetterPosition)
{
  uint8_t Height = size * 8;
  uint8_t Width = size * 6;
  tft.setCursor(Width * LetterPosition, Height * LineNumber);
}

//Draws Smith Chart
void DrawSmithChart()
{
  tft.drawCircle(240, 160, 160, BLACK); //Draw Smith Chart circle
  tft.drawCircle(320, 160, 80, BLACK);  //Draw left circle
  tft.drawCircle(160, 160, 80, BLACK);  //Draw right circle
  tft.drawCircle(240, 160, 53, GREEN);  //2:1 SWR Circle
  tft.drawCircle(240, 160, 80, RED);  //3:1 SWR Circle
  tft.drawFastVLine(240, 0, 320, BLACK);  //Draw vertical line
  tft.drawFastHLine(80, 160, 320, BLACK); //Draw horizontal line
}

// Graphics speed test
void GraphicsSpeedTest()
{
//  tft.drawCircle(120, 120, 3, BLACK); //Plot value
tft.drawRect(120,120,6, 6,BLACK);
}

//SWRCalc calculates the SWR
void SWRCalc()
{
  float SWR;
  SWR = (1 + abs(GammaMag)) / (1 - abs(GammaMag));
  tft.setTextColor(RED);
  tft.setTextSize(2);
  tft.setCursor(5, 300);
  tft.print("SWR = ");
  if(SWR > 999) {SWR = 999;}
  tft.print(SWR, 2);
}

//ZPrint //Prints the impedance
void ZPrint()
{

   //Print results
  tft.setTextColor(RED);
  tft.setTextSize(2);
  tft.setCursor(330, 300);
  tft.print("Z = ");
  if(ZReal > 10000) {ZReal = 10000;}
  tft.print(int(ZReal));
  if(ZIm >= 0)
  {
    tft.print("+J");
  }
  else
  {
    tft.print("-J");
  }
```

```
  if(ZIm > 10000) {ZIm = 10000;}
  tft.print(int(abs(ZIm)));
}

void TouchScreenSelect()
{
 TSPoint p = ts.getPoint();

    if((p.x == 0) && (p.y == 0) && (p.z == 0))
    {
      Band = 80;  //Band = 80 meters (3.74 MHz).
      Label_40m_Red(); digitalWrite(Relay40, LOW); //Open 40m 90 degree phase shift circuit
      Label_80m_Green(); digitalWrite(Relay80, HIGH); //Close 80m 90 degree phase shift circuit
      Label_20m_Red(); digitalWrite(Relay20, LOW); //Open 20m 90 degree phase shift circuit
    }

    else if((p.x > 300) && (p.y < 300))  //Switch to 40 meters
    {
      Band = 40;  //Band = 40 meters (7.148 MHz)
      Label_40m_Green(); digitalWrite(Relay40, HIGH); //Close 40m 90 degree phase shift circuit
      Label_80m_Red(); digitalWrite(Relay80, LOW); //Open 80m 90 degree phase shift circuit
      Label_20m_Red(); digitalWrite(Relay20, LOW); //Open 20m 90 degree phase shift circuit
    }
    else if((p.x > 500) && (p.y > 600))
    {
      Band = 20;  //Band = 20 meters (14.174 MHz)
      Label_40m_Red(); digitalWrite(Relay40, LOW); //Open 40m 90 degree phase shift circuit
      Label_80m_Red(); digitalWrite(Relay80, LOW); //Open 80m 90 degree phase shift circuit
      Label_20m_Green(); digitalWrite(Relay20, HIGH); //Close 20m 90 degree phase shift circuit
    }

}

void Label_20m_Green()
{
  TextParameters(3, GREEN, WHITE);  //Set
  CharacterCursorPosition(3, 1, 23);
  tft.print("20m");
}

void Label_40m_Green()
{
  TextParameters(3, GREEN, WHITE);  //Set
  CharacterCursorPosition(3, 1, 1);
  tft.print("40m");
}

void Label_80m_Green()
{
  TextParameters(3, GREEN, WHITE);  //Set
  CharacterCursorPosition(3, 10, 1);
  tft.print("80m");
}



void Label_20m_Red()
{
  TextParameters(3, RED, WHITE);  //Set
  CharacterCursorPosition(3, 1, 23);
  tft.print("20m");
}

void Label_40m_Red()
{
  TextParameters(3, RED, WHITE);  //Set
  CharacterCursorPosition(3, 1, 1);
  tft.print("40m");
}

void Label_80m_Red()
{
  TextParameters(3, RED, WHITE);  //Set
  CharacterCursorPosition(3, 10, 1);
  tft.print("80m");
}

void TouchScreenTest()
{
 TSPoint p = ts.getPoint();
 if(p.z > ts.pressureThreshhold)
  {
```

```
    Xcordinate = p.x;
    Ycordinate = p.y;
  }
//Serial.print(Xcordinate);
//Serial.print("  ");
//Serial.println(Ycordinate);
}
```