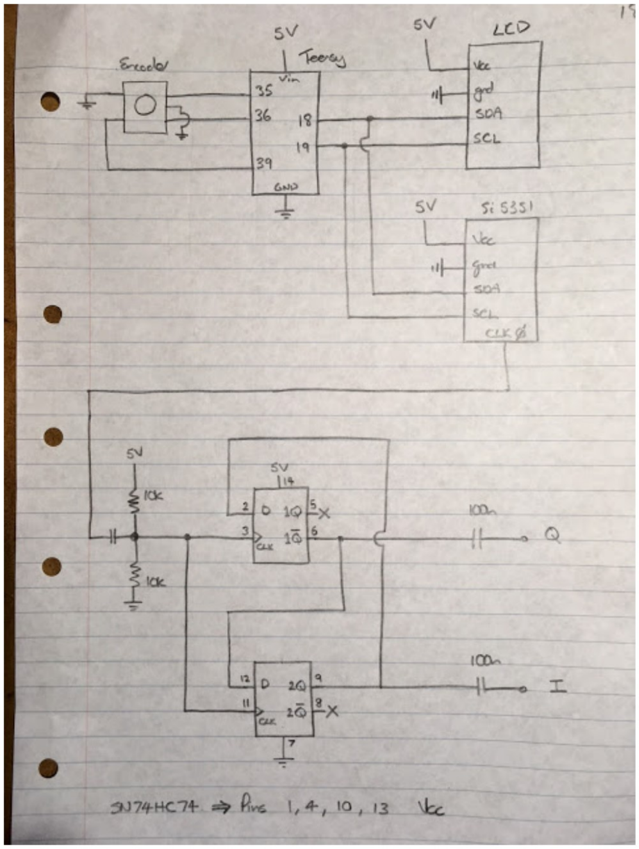
Charlie Morris, ZL2CTM

Amateur Radio Homebrew Experiments

Tuesday, 20 March 2018

Homebrew SSB SDR Rig

Part 1. Quadrature Oscillator. See my ZL2CTM YouTube channel for accompanying video.



Code. This will be updated as additional SDR DSP functions are added.

```
#include <Wire.h>                          // I2C comms
#include <si5351.h>                         // Si5351 library
#include <LiquidCrystal_I2C.h>            // LCD library


// Define Constants and Vaviables
static const long bandStart = 1000000;     // start of HF band
static const long bandEnd =   30000000;    // end of HF band
static const long bandInit =  3690000;     // where to initially set the frequency
volatile long oldfreq = 0;
volatile long freq = bandInit ;
volatile long radix = 1000;                // how much to change the frequency by, clicking the rotary
encoder will change this.
volatile int updatedisplay = 0;

// Rotary Encoder
static const int pushPin = 39;
static const int rotBPin = 36;
static const int rotAPin = 35;
volatile int rotState = 0;
```

About Me

Charlie Morris

View my complete profile

Blog Archive

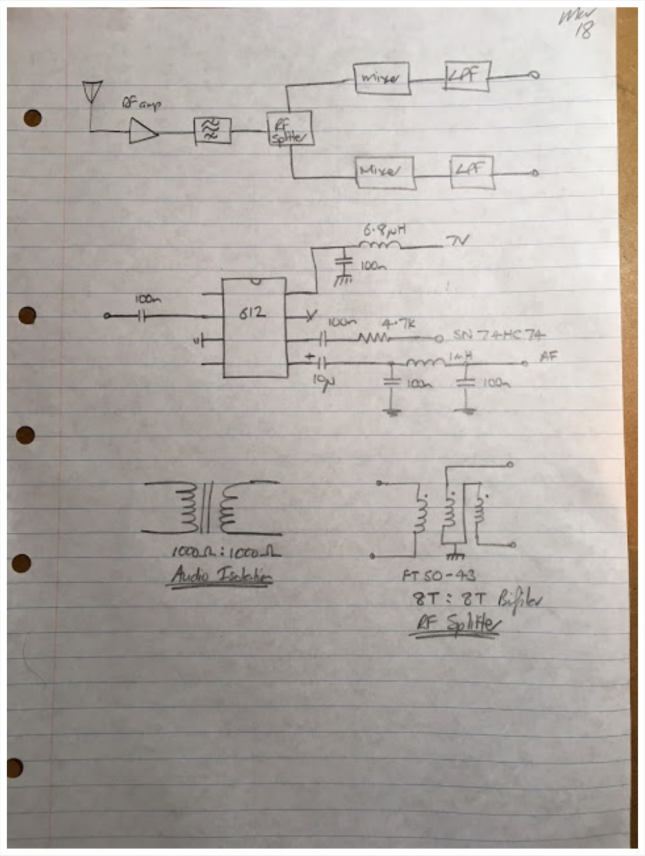
- ▼ 2018 (5)
  - ► August (1)
  - ► June (1)
  - ▼ March (2)
    - Homebrew SSB SDR Rig
    - Si5351 DDS VFO/BFO Example Software
  - ► January (1)
- ► 2017 (5)

```
volatile int rotAval = 1;
volatile int rotBval = 1;
volatile int rotAcc = 0;

// Instantiate the Objects
LiquidCrystal_I2C lcd(0x3F, 16, 2);       // set the LCD address to either 0x27 or 0x3F for a 16
chars and 2 line display
Si5351 si5351;


void setup()
{
  // Set up input switches
  pinMode(rotAPin, INPUT);
  pinMode(rotBPin, INPUT);
  pinMode(pushPin, INPUT);
  digitalWrite(rotAPin, HIGH);
  digitalWrite(rotBPin, HIGH);
  digitalWrite(pushPin, HIGH);

  // Set up interrupt pins
  attachInterrupt(digitalPinToInterrupt(rotAPin), ISRrotAChange, CHANGE);
  attachInterrupt(digitalPinToInterrupt(rotBPin), ISRrotBChange, CHANGE);

  // Initialise the lcd
  lcd.begin();
  lcd.backlight();

  // Initialise the DDS
  si5351.init(SI5351_CRYSTAL_LOAD_8PF, 0);
  si5351.set_pll(SI5351_PLL_FIXED, SI5351_PLLA);
  si5351.drive_strength(SI5351_CLK0, SI5351_DRIVE_8MA);
  si5351.set_freq((freq * 100ULL), SI5351_PLL_FIXED, SI5351_CLK0);

  UpdateDisplay();
}


void loop()
{
  if (freq != oldfreq)
  {
    UpdateDisplay();
    SendFrequency();
    oldfreq = freq;
  }

  if (digitalRead(pushPin) == LOW)
  {
    delay(10);
    while (digitalRead(pushPin) == LOW)
    {
      if (updatedisplay == 1)
      {
        UpdateDisplay();
        updatedisplay = 0;
      }
    }
    delay(50);
  }
}


// Interrupt routines
void ISRrotAChange()
{
  if (digitalRead(rotAPin))
  {
    rotAval = 1;
    UpdateRot();
  }
  else
  {
    rotAval = 0;
    UpdateRot();
  }
}
```

```
void ISRrotBChange()
{
  if (digitalRead(rotBPin))
  {
    rotBval = 1;
    UpdateRot();
  }
  else
  {
    rotBval = 0;
    UpdateRot();
  }
}


void UpdateRot()
{
  switch (rotState)
  {

    case 0:                                // Idle state, look for direction
      if (!rotBval)
        rotState = 1;                        // CW 1
      if (!rotAval)
        rotState = 11;                        // CCW 1
      break;

    case 1:                                // CW, wait for A low while B is low
      if (!rotBval)
      {
        if (!rotAval)
        {
          // either increment radixindex or freq
          if (digitalRead(pushPin) == LOW)
          {
            updatedisplay = 1;
            if (radix == 1000000)
              radix = 100000;
            else if (radix == 100000)
              radix = 10000;
            else if (radix == 10000)
              radix = 1000;
            else if (radix == 1000)
              radix = 100;
            else if (radix == 100)
              radix = 10;
            else if (radix == 10)
              radix = 1;
            else
              radix = 1000000;
          }
          else
          {
            freq = (freq + radix);
            if (freq > bandEnd)
              freq = bandEnd;
          }
          rotState = 2;                      // CW 2
        }
      }
      else if (rotAval)
        rotState = 0;                        // It was just a glitch on B, go back to start
      break;

    case 2:                                // CW, wait for B high
      if (rotBval)
        rotState = 3;                        // CW 3
      break;

    case 3:                                // CW, wait for A high
      if (rotAval)
        rotState = 0;                        // back to idle (detent) state
      break;

    case 11:                                // CCW, wait for B low while A is low
      if (!rotAval)
      {
        if (!rotBval)
        {
          // either decrement radixindex or freq
          if (digitalRead(pushPin) == LOW)
          {
            updatedisplay = 1;
            if (radix == 1)
              radix = 10;
            else if (radix == 10)
              radix = 100;
            else if (radix == 100)
              radix = 1000;
            else if (radix == 1000)
              radix = 10000;
            else if (radix == 10000)
              radix = 100000;
            else if (radix == 100000)
              radix = 1000000;
            else
              radix = 1;
          }
          else
          {
            freq = (freq - radix);
            if (freq < bandStart)
              freq = bandStart;
          }
          rotState = 12;                            // CCW 2
        }
      }
      else if (rotBval)
        rotState = 0;                              // It was just a glitch on A, go back to start
      break;

    case 12:                                      // CCW, wait for A high
      if (rotAval)
        rotState = 13;                            // CCW 3
      break;

    case 13:                                      // CCW, wait for B high
      if (rotBval)
        rotState = 0;                              // back to idle (detent) state
      break;
  }
}


void UpdateDisplay()
{
  lcd.cursor();                                   // Turn on the cursor
  lcd.setCursor(0, 0);
  lcd.print("        ");
  lcd.setCursor(0, 0);
  lcd.print(freq);
  lcd.setCursor(10, 0);
  lcd.print("ZL2CTM");

  lcd.setCursor(0, 1);
  lcd.print("        ");
  lcd.setCursor(0, 1);

  if (freq > 9999999)
  {
    if (radix == 1)
      lcd.setCursor(7, 0);
    if (radix == 10)
      lcd.setCursor(6, 0);
    if (radix == 100)
      lcd.setCursor(5, 0);
    if (radix == 1000)
      lcd.setCursor(4, 0);
    if (radix == 10000)
      lcd.setCursor(3, 0);
    if (radix == 100000)
      lcd.setCursor(2, 0);
    if (radix == 1000000)
      lcd.setCursor(1, 0);

  }
  if (freq <= 9999999)
  {
    if (radix == 1)
      lcd.setCursor(6, 0);
    if (radix == 10)
      lcd.setCursor(5, 0);
    if (radix == 100)
      lcd.setCursor(4, 0);
    if (radix == 1000)
      lcd.setCursor(3, 0);
    if (radix == 10000)
      lcd.setCursor(2, 0);
    if (radix == 100000)
      lcd.setCursor(1, 0);
    if (radix == 1000000)
      lcd.setCursor(0, 0);
  }
}



void SendFrequency()
{
  si5351.set_freq((freq * 4) * 100ULL, SI5351_PLL_FIXED, SI5351_CLK0);
}
```

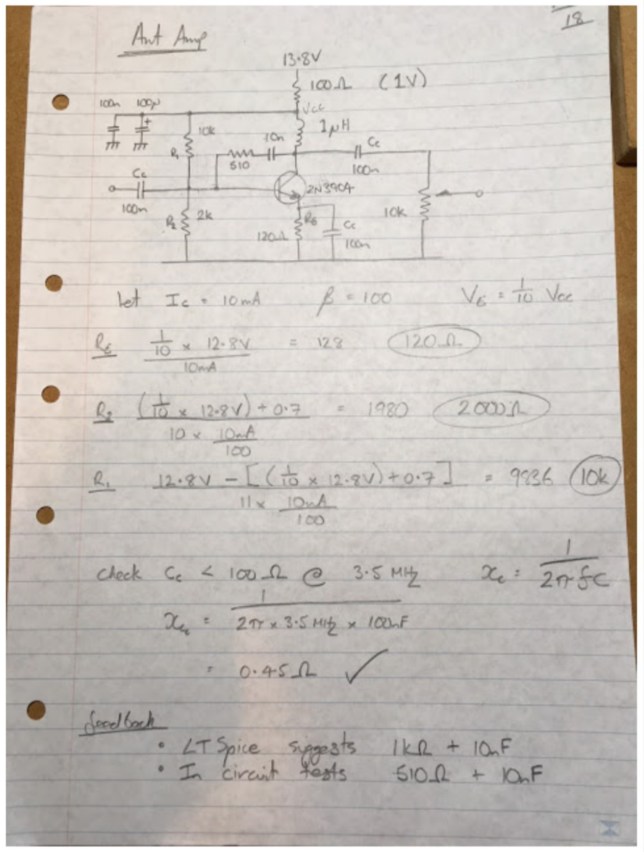
****

Dual Quadrature NE612 Direct Conversion Front End



Antenna RF Amplifier

Please note the collector inductor below is 1mH NOT 1uH



Ant Amp

let $I_c = 10mA$ $\beta = 100$ $V_E = \frac{1}{10} V_{cc}$

R_E $\frac{\frac{1}{10} \times 12{\cdot}8V}{10mA} = 128$ $\boxed{120\Omega}$

R_2 $\frac{(\frac{1}{10} \times 12{\cdot}8V) + 0{\cdot}7}{10 \times \frac{10mA}{100}} = 1980$ $\boxed{2000\Omega}$

R_1 $\frac{12{\cdot}8V - [(\frac{1}{10} \times 12{\cdot}8V) + 0{\cdot}7]}{11 \times \frac{10mA}{100}} = 9936$ $\boxed{10k}$

Check $C_c < 100\Omega$ @ 3·5 MHz $X_c = \frac{1}{2\pi f C}$

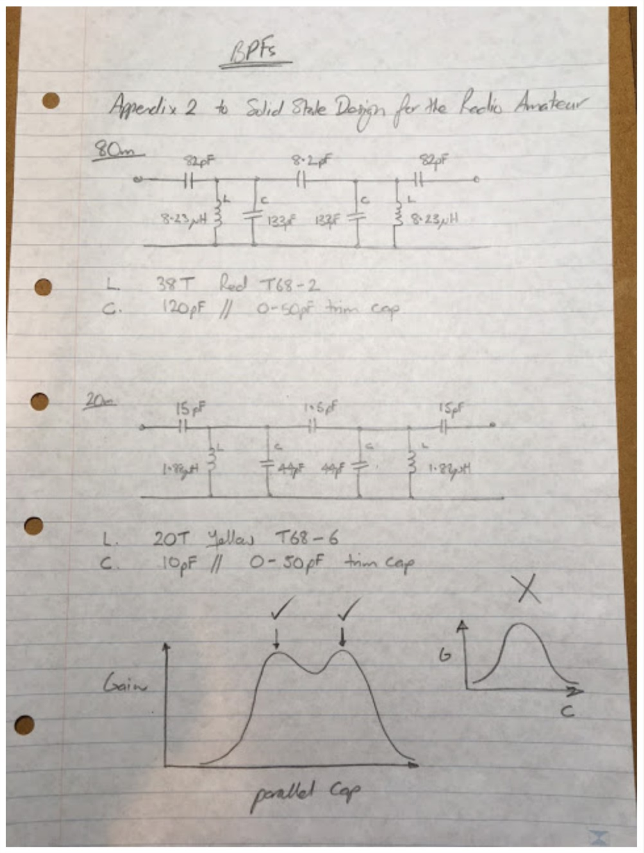
$X_{c_c} = \frac{1}{2\pi \times 3{\cdot}5 MHz \times 100nF}$

$= 0{\cdot}45\Omega$ ✓

Feedback

- LT Spice suggests 1kΩ + 10nF
- In circuit tests 510Ω + 10nF

80m/20m Bandpass Filter



BPFs

Appendix 2 to Solid State Design for the Radio Amateur

80m

L. 38T Red T68-2
C. 120pF // 0-50pF trim cap

20m

L. 20T Yellow T68-6
C. 10pF // 0-50pF trim cap

Audio Pass-through Test Code

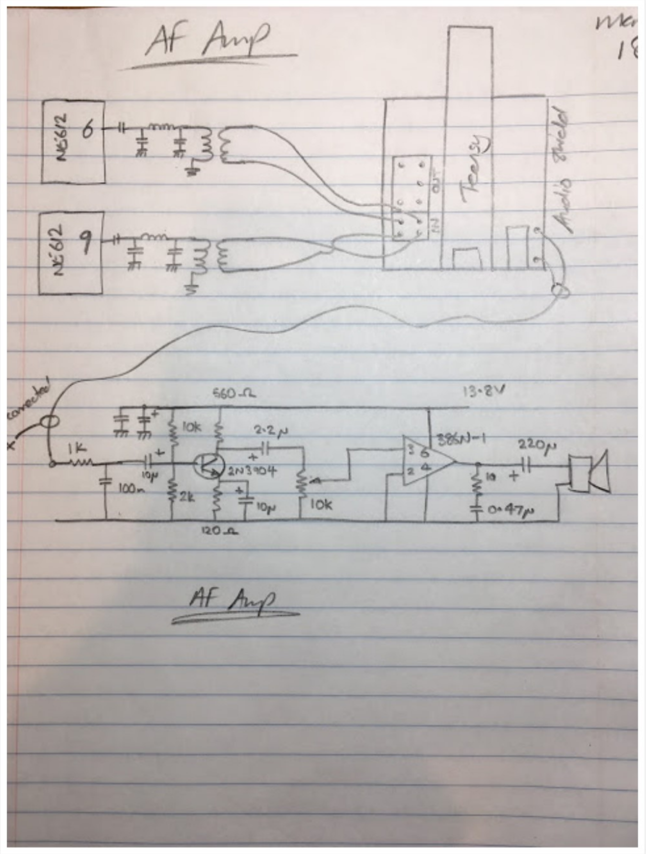
```
#include <Wire.h>                          // I2C comms
#include <si5351.h>                         // Si5351 library
#include <LiquidCrystal_I2C.h>              // LCD library
#include <Audio.h>                          // Teensy audio library

// Define Constants and Vaviables
static const long bandStart = 1000000;     // start of HF band
static const long bandEnd =   30000000;    // end of HF band
static const long bandInit =  3690000;     // where to initially set the frequency
volatile long oldfreq = 0;
volatile long freq = bandInit ;
volatile long radix = 1000;                // how much to change the frequency by, clicking the rotary
encoder will change this.
volatile int updatedisplay = 0;

// Rotary Encoder
static const int pushPin = 39;
static const int rotBPin = 36;
static const int rotAPin = 35;
volatile int rotState = 0;
volatile int rotAval = 1;
volatile int rotBval = 1;
volatile int rotAcc = 0;

// Instantiate the Objects
LiquidCrystal_I2C lcd(0x3F, 16, 2);       // Set the LCD address to either 0x27 or 0x3F for a 16
chars and 2 line display
Si5351 si5351;                            // The Si5351 DDS
AudioControlSGTL5000    audioShield;      // The Teensy audio CODEC on the audio shield


// Audio shield
AudioInputI2S          audioInput;                                   // What we call the input to the audio shield
AudioOutputI2S          audioOutput;                                 // What we call the output of the audio
shield
AudioConnection         patchCord5(audioInput, 0, audioOutput, 0);    // Left channel in to left
channel out
AudioConnection         patchCord10(audioInput, 1, audioOutput, 1);   // Right channel in to right
channel out

void setup()
{
  // Setup input switches
  pinMode(rotAPin, INPUT);
  pinMode(rotBPin, INPUT);
  pinMode(pushPin, INPUT);
  digitalWrite(rotAPin, HIGH);
  digitalWrite(rotBPin, HIGH);
  digitalWrite(pushPin, HIGH);

  // Setup interrupt pins
  attachInterrupt(digitalPinToInterrupt(rotAPin), ISRrotAChange, CHANGE);
  attachInterrupt(digitalPinToInterrupt(rotBPin), ISRrotBChange, CHANGE);

  // Setup the lcd
  lcd.begin();
  lcd.backlight();

  // Setup the DDS
  si5351.init(SI5351_CRYSTAL_LOAD_8PF, 0);
  si5351.set_pll(SI5351_PLL_FIXED, SI5351_PLLA);
  si5351.drive_strength(SI5351_CLK0, SI5351_DRIVE_8MA);
  si5351.set_freq((freq * 100ULL), SI5351_PLL_FIXED, SI5351_CLK0);

  // Setup the audio shield
  AudioNoInterrupts();
  AudioMemory(16);
  audioShield.enable();
  audioShield.inputSelect(AUDIO_INPUT_LINEIN);
  audioShield.volume(0.7);
  audioShield.unmuteLineout();
  AudioInterrupts();

  UpdateDisplay();
}


void loop()
```

```
{
  if (freq != oldfreq)
  {
    UpdateDisplay();
    SendFrequency();
    oldfreq = freq;
  }

  if (digitalRead(pushPin) == LOW)
  {
    delay(10);
    while (digitalRead(pushPin) == LOW)
    {
      if (updatedisplay == 1)
      {
        UpdateDisplay();
        updatedisplay = 0;
      }
    }
    delay(50);
  }
}


// Interrupt routines
void ISRrotAChange()
{
  if (digitalRead(rotAPin))
  {
    rotAval = 1;
    UpdateRot();
  }
  else
  {
    rotAval = 0;
    UpdateRot();
  }
}


void ISRrotBChange()
{
  if (digitalRead(rotBPin))
  {
    rotBval = 1;
    UpdateRot();
  }
  else
  {
    rotBval = 0;
    UpdateRot();
  }
}


void UpdateRot()
{
  switch (rotState)
  {

    case 0:                                          // Idle state, look for direction
      if (!rotBval)
        rotState = 1;                                  // CW 1
      if (!rotAval)
        rotState = 11;                                  // CCW 1
      break;

    case 1:                                          // CW, wait for A low while B is low
      if (!rotBval)
      {
        if (!rotAval)
        {
          // either increment radixindex or freq
          if (digitalRead(pushPin) == LOW)
          {
            updatedisplay = 1;
            if (radix == 1000000)
              radix = 100000;
            else if (radix == 100000)
              radix = 10000;
            else if (radix == 10000)
              radix = 1000;
            else if (radix == 1000)
              radix = 100;
            else if (radix == 100)
              radix = 10;
            else if (radix == 10)
              radix = 1;
            else
              radix = 1000000;
          }
          else
          {
            freq = (freq + radix);
            if (freq > bandEnd)
              freq = bandEnd;
          }
          rotState = 2;                    // CW 2
        }
      }
      else if (rotAval)
        rotState = 0;                      // It was just a glitch on B, go back to start
      break;

    case 2:                                // CW, wait for B high
      if (rotBval)
        rotState = 3;                      // CW 3
      break;

    case 3:                                // CW, wait for A high
      if (rotAval)
        rotState = 0;                      // back to idle (detent) state
      break;

    case 11:                               // CCW, wait for B low while A is low
      if (!rotAval)
      {
        if (!rotBval)
        {
          // either decrement radixindex or freq
          if (digitalRead(pushPin) == LOW)
          {
            updatedisplay = 1;
            if (radix == 1)
              radix = 10;
            else if (radix == 10)
              radix = 100;
            else if (radix == 100)
              radix = 1000;
            else if (radix == 1000)
              radix = 10000;
            else if (radix == 10000)
              radix = 100000;
            else if (radix == 100000)
              radix = 1000000;
            else
              radix = 1;
          }
          else
          {
            freq = (freq - radix);
            if (freq < bandStart)
              freq = bandStart;
          }
          rotState = 12;                   // CCW 2
        }
      }
      else if (rotBval)
        rotState = 0;                      // It was just a glitch on A, go back to start
      break;

    case 12:                               // CCW, wait for A high
      if (rotAval)
        rotState = 13;                     // CCW 3
      break;

    case 13:                               // CCW, wait for B high
      if (rotBval)
```

```
        rotState = 0;                               // back to idle (detent) state
      break;
  }
}


void UpdateDisplay()
{
  lcd.cursor();                                      // Turn on the cursor
  lcd.setCursor(0, 0);
  lcd.print("        ");
  lcd.setCursor(0, 0);
  lcd.print(freq);
  lcd.setCursor(10, 0);
  lcd.print("ZL2CTM");

  lcd.setCursor(0, 1);
  lcd.print("        ");
  lcd.setCursor(0, 1);

  if (freq > 9999999)
  {
    if (radix == 1)
      lcd.setCursor(7, 0);
    if (radix == 10)
      lcd.setCursor(6, 0);
    if (radix == 100)
      lcd.setCursor(5, 0);
    if (radix == 1000)
      lcd.setCursor(4, 0);
    if (radix == 10000)
      lcd.setCursor(3, 0);
    if (radix == 100000)
      lcd.setCursor(2, 0);
    if (radix == 1000000)
      lcd.setCursor(1, 0);

  }
  if (freq <= 9999999)
  {
    if (radix == 1)
      lcd.setCursor(6, 0);
    if (radix == 10)
      lcd.setCursor(5, 0);
    if (radix == 100)
      lcd.setCursor(4, 0);
    if (radix == 1000)
      lcd.setCursor(3, 0);
    if (radix == 10000)
      lcd.setCursor(2, 0);
    if (radix == 100000)
      lcd.setCursor(1, 0);
    if (radix == 1000000)
      lcd.setCursor(0, 0);
  }
}



void SendFrequency()
{
  si5351.set_freq((freq * 4) * 100ULL, SI5351_PLL_FIXED, SI5351_CLK0);
}
```

****

AF Amplifier



Receive Test Code

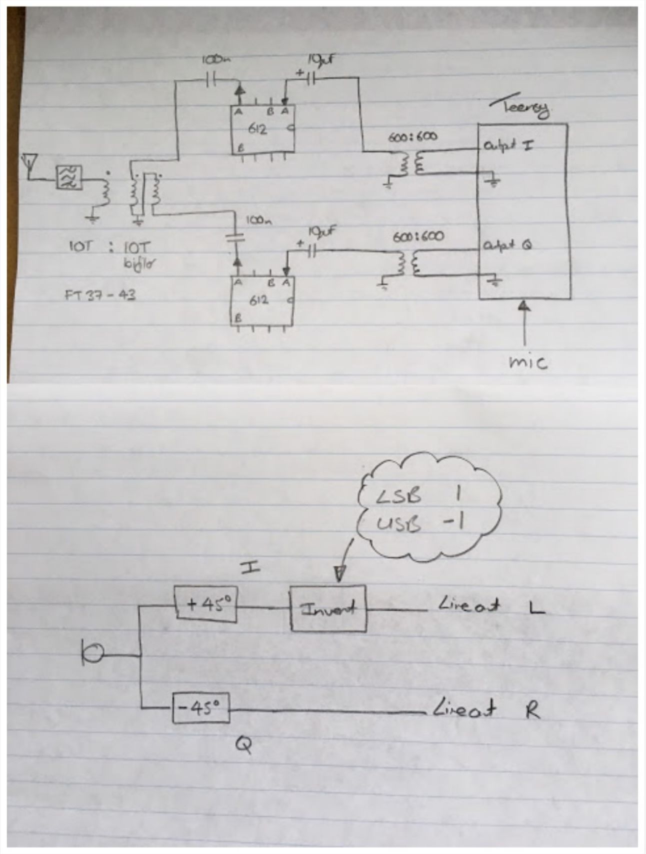
```
// Libraries
#include <Wire.h>                          // I2C comms library
#include <si5351.h>                        // Si5351Jason library
#include <LiquidCrystal_I2C.h>             // LCD library
#include <Audio.h>                         // Teensy audio library

// Number of Filter Coefficients
#define NO_HILBERT_COEFFS 70               // Used to define the Hilbert transform filter arrays.
More typical than 'const int'.

// Define Constants and Vaviables
static const long bandStart = 1000000;     // start of HF band
static const long bandEnd =   30000000;    // end of HF band
static const long bandInit =  3690000;     // where to initially set the frequency
volatile long oldfreq = 0;
volatile long freq = bandInit ;
volatile long radix = 1000;                // how much to change the frequency by clicking the rotary
encoder will change this.
volatile int updatedisplay = 0;

// Rotary Encoder
static const int pushPin = 39;
static const int rotBPin = 36;
static const int rotAPin = 35;
volatile int rotState = 0;
volatile int rotAval = 1;
volatile int rotBval = 1;
volatile int rotAcc = 0;


// Iowa Hills Hilbert transform filter coefficients
const short Hilbert_Plus_45_Coeffs[NO_HILBERT_COEFFS] = {
  (short)(32768 * -0.000287988910943357),
  (short)(32768 * -0.000383511439791303),
  (short)(32768 * -0.000468041804899774),
  (short)(32768 * -0.000529324432676899),
  (short)(32768 * -0.000569479602046985),
  (short)(32768 * -0.000616670267768531),
  (short)(32768 * -0.000731530748681977),
  (short)(32768 * -0.001002372095321225),
  (short)(32768 * -0.001525299390682192),
  (short)(32768 * -0.002370114347025230),
  (short)(32768 * -0.003539247773172147),
  (short)(32768 * -0.004932965382552984),
  (short)(32768 * -0.006337182914262393),
  (short)(32768 * -0.007448193692118567),
  (short)(32768 * -0.007940501940620482),
  (short)(32768 * -0.007570802072162988),
  (short)(32768 * -0.006296120449841751),
  (short)(32768 * -0.004371955618154949),
  (short)(32768 * -0.002391875073164555),
  (short)(32768 * -0.001236984700413469),
  (short)(32768 * -0.001922560128827416),
  (short)(32768 * -0.005356720327533458),
  (short)(32768 * -0.012055656297010635),
  (short)(32768 * -0.021882952959947619),
  (short)(32768 * -0.033888748300090733),
  (short)(32768 * -0.046312736456333638),
  (short)(32768 * -0.056783367797647665),
  (short)(32768 * -0.062699937453677912),
  (short)(32768 * -0.061735375084135742),
  (short)(32768 * -0.052358513976237808),
  (short)(32768 * -0.034257179158167443),
  (short)(32768 * -0.008554500746482946),
  (short)(32768 * 0.022249911747384360),
  (short)(32768 * 0.054622962942346594),
  (short)(32768 * 0.084568844473140448),
  (short)(32768 * 0.108316122839950818),
  (short)(32768 * 0.122979341462627859),
  (short)(32768 * 0.127056096658453188),
  (short)(32768 * 0.120656295327679283),
  (short)(32768 * 0.105420364259485699),
  (short)(32768 * 0.084152608145489444),
  (short)(32768 * 0.060257510644444748),
  (short)(32768 * 0.037105711921879434),
  (short)(32768 * 0.017464092086704748),
  (short)(32768 * 0.003100559033325746),
  (short)(32768 * -0.005373489802481697),
  (short)(32768 * -0.008418211280310166),
  (short)(32768 * -0.007286730644726664),
  (short)(32768 * -0.003638388931163832),
  (short)(32768 * 0.000858330713630433),
  (short)(32768 * 0.004847436504682235),
  (short)(32768 * 0.007476399317750315),
  (short)(32768 * 0.008440227567663121),
  (short)(32768 * 0.007898970420636600),
  (short)(32768 * 0.006314366257036837),
  (short)(32768 * 0.004261033495040515),
  (short)(32768 * 0.002261843500794377),
  (short)(32768 * 0.000680212977485724),
  (short)(32768 * -0.000319493110301691),
  (short)(32768 * -0.000751893569425181),
  (short)(32768 * -0.000752248417868501),
  (short)(32768 * -0.000505487955986662),
  (short)(32768 * -0.000184645628631330),
  (short)(32768 * 0.000087913008490067),
  (short)(32768 * 0.000253106348867209),
  (short)(32768 * 0.000306473486382603),
  (short)(32768 * 0.000277637042003169),
  (short)(32768 * 0.000207782317481292),
  (short)(32768 * 0.000132446796990356),
  (short)(32768 * 0.000072894261560354)
};

// Iowa Hills Hilbert transform filter coefficients
const short Hilbert_Minus_45_Coeffs[NO_HILBERT_COEFFS] = {
  (short)(32768 * -0.000072894261560345),
  (short)(32768 * -0.000132446796990344),
  (short)(32768 * -0.000207782317481281),
  (short)(32768 * -0.000277637042003168),
  (short)(32768 * -0.000306473486382623),
  (short)(32768 * -0.000253106348867259),
  (short)(32768 * -0.000087913008490148),
  (short)(32768 * 0.000184645628631233),
  (short)(32768 * 0.000505487955986583),
  (short)(32768 * 0.000752248417868491),
  (short)(32768 * 0.000751893569425298),
  (short)(32768 * 0.000319493110301983),
  (short)(32768 * -0.000680212977485245),
  (short)(32768 * -0.002261843500793748),
  (short)(32768 * -0.004261033495039842),
  (short)(32768 * -0.006314366257036280),
  (short)(32768 * -0.007898970420636345),
  (short)(32768 * -0.008440227567663343),
  (short)(32768 * -0.007476399317751102),
  (short)(32768 * -0.004847436504683540),
  (short)(32768 * -0.000858330713632029),
  (short)(32768 * 0.003638388931162351),
  (short)(32768 * 0.007286730644725833),
  (short)(32768 * 0.008418211280310565),
  (short)(32768 * 0.005373489802483816),
  (short)(32768 * -0.003100559033321630),
  (short)(32768 * -0.017464092086698697),
  (short)(32768 * -0.037105711921871905),
  (short)(32768 * -0.060257510644436532),
  (short)(32768 * -0.084152608145481672),
  (short)(32768 * -0.105420364259479538),
  (short)(32768 * -0.120656295327675800),
  (short)(32768 * -0.127056096658453216),
  (short)(32768 * -0.122979341462631633),
  (short)(32768 * -0.108316122839958146),
  (short)(32768 * -0.084568844473150454),
  (short)(32768 * -0.054622962942358168),
  (short)(32768 * -0.022249911747396132),
  (short)(32768 * 0.008554500746472333),
  (short)(32768 * 0.034257179158159054),
  (short)(32768 * 0.052358513976232306),
  (short)(32768 * 0.061735375084133286),
  (short)(32768 * 0.062699937453678217),
  (short)(32768 * 0.056783367797650072),
  (short)(32768 * 0.046312736456337288),
  (short)(32768 * 0.033888748300094730),
  (short)(32768 * 0.021882952959951244),
  (short)(32768 * 0.012055656297013388),
  (short)(32768 * 0.005356720327535105),
  (short)(32768 * 0.001922560128828006),
  (short)(32768 * 0.001236984700413229),
  (short)(32768 * 0.002391875073163812),
  (short)(32768 * 0.004371955618154038),
  (short)(32768 * 0.006296120449840938),
  (short)(32768 * 0.007570802072162439),
  (short)(32768 * 0.007940501940620253),
  (short)(32768 * 0.007448193692118624),
  (short)(32768 * 0.006337182914262643),
  (short)(32768 * 0.004932965382553323),
  (short)(32768 * 0.003539247773172483),
  (short)(32768 * 0.002370114347025498),
  (short)(32768 * 0.001525299390682370),
  (short)(32768 * 0.001002372095321316),
  (short)(32768 * 0.000731530748682004),
  (short)(32768 * 0.000616670267768521),
  (short)(32768 * 0.000569479602046963),
  (short)(32768 * 0.000529324432676881),
  (short)(32768 * 0.000468041804899765),
  (short)(32768 * 0.000383511439791304),
  (short)(32768 * 0.000287988910943362)
};


// Instantiate the Objects
LiquidCrystal_I2C lcd(0x3F, 16, 2);       // Name for the LCD. Set the LCD address to either 0x27
or 0x3F for a 16 chars and 2 line display
Si5351 si5351;                            // Name for the Si5351 DDS
AudioControlSGTL5000    audioShield;      // Name for the Teensy audio CODEC on the audio
shield

// Audio shield
AudioInputI2S           audioInput;                                       // Name for the input to the audio
shield
AudioOutputI2S          audioOutput;                                      // Name for the output of the audio
shield
// Receiver
AudioFilterFIR          RX_Hilbert_Plus_45;                               // Name for the RX +45 Hilbert
transform
AudioFilterFIR          RX_Hilbert_Minus_45;                              // Name for the RX +45 Hilbert
transform
AudioMixer4             RX_Summer;                                        // Name for the RX summer
```

```
// Audio connections
AudioConnection         patchCord5(audioInput, 0, RX_Hilbert_Plus_45, 0);     // Left channel in
Hilbert transform +45
AudioConnection         patchCord10(audioInput, 1, RX_Hilbert_Minus_45, 0);   // Right channel in
Hilbert transform -45
AudioConnection         patchCord15(RX_Hilbert_Plus_45, 0, RX_Summer, 0);     // Hilbert
transform +45 to receiver summer
AudioConnection         patchCord20(RX_Hilbert_Minus_45, 0, RX_Summer, 1);    // Hilbert
transform -45 to receiver summer
AudioConnection         patchCord25(RX_Summer, 0, audioOutput, 0);            // Receiver summer
to receiver LPF


void setup()
{
  // Setup input switches
  pinMode(rotAPin, INPUT);
  pinMode(rotBPin, INPUT);
  pinMode(pushPin, INPUT);
  digitalWrite(rotAPin, HIGH);
  digitalWrite(rotBPin, HIGH);
  digitalWrite(pushPin, HIGH);

  // Setup interrupt pins
  attachInterrupt(digitalPinToInterrupt(rotAPin), ISRrotAChange, CHANGE);
  attachInterrupt(digitalPinToInterrupt(rotBPin), ISRrotBChange, CHANGE);

  // Setup the lcd
  lcd.begin();
  lcd.backlight();

  // Setup the DDS
  si5351.init(SI5351_CRYSTAL_LOAD_8PF, 0);
  si5351.set_pll(SI5351_PLL_FIXED, SI5351_PLLA);
  si5351.drive_strength(SI5351_CLK0, SI5351_DRIVE_8MA);
  si5351.set_freq((freq * 100ULL), SI5351_PLL_FIXED, SI5351_CLK0);

  // Setup the audio shield
  AudioNoInterrupts();
  AudioMemory(16);
  audioShield.enable();
  audioShield.volume(0.7);            // Constant. Use external volume control on the audio amp
  AudioInterrupts();

  // Setup transceiver mode
  Turn_On_Receiver();
  UpdateDisplay();
}

void loop()
{
  if (freq != oldfreq)                // Check to see if the frequency has changed. If so, update
everything.
  {
    UpdateDisplay();
    SendFrequency();
    oldfreq = freq;
  }

  if (digitalRead(pushPin) == LOW)    // Update cursor, but also stop it from flickering
  {
    delay(10);
    while (digitalRead(pushPin) == LOW)
    {
      if (updatedisplay == 1)
      {
        UpdateDisplay();
        updatedisplay = 0;
      }
    }
    delay(50);
  }
}


void Turn_On_Receiver()
{
  AudioNoInterrupts();
  audioShield.inputSelect(AUDIO_INPUT_LINEIN);
  audioShield.lineInLevel(5);                                             // Default is 5
  audioShield.unmuteHeadphone();
  RX_Hilbert_Plus_45.begin(Hilbert_Plus_45_Coeffs, NO_HILBERT_COEFFS);
  RX_Hilbert_Minus_45.begin(Hilbert_Minus_45_Coeffs, NO_HILBERT_COEFFS);

  if (freq <= 9999999)          // LSB
  {
    RX_Summer.gain(0, 1);
    RX_Summer.gain(1, -1);
  }
  if (freq > 9999999)           // USB
  {
    RX_Summer.gain(0, 1);
    RX_Summer.gain(1, 1);
  }

  AudioInterrupts();
}


// Interrupt routines
void ISRrotAChange()
{
  if (digitalRead(rotAPin))
  {
    rotAval = 1;
    UpdateRot();
  }
  else
  {
    rotAval = 0;
    UpdateRot();
  }
}


void ISRrotBChange()
{
  if (digitalRead(rotBPin))
  {
    rotBval = 1;
    UpdateRot();
  }
  else
  {
    rotBval = 0;
    UpdateRot();
  }
}


void UpdateRot()
{
  switch (rotState)
  {

    case 0:                                         // Idle state, look for direction
      if (!rotBval)
        rotState = 1;                               // CW 1
      if (!rotAval)
        rotState = 11;                              // CCW 1
      break;

    case 1:                                         // CW, wait for A low while B is low
      if (!rotBval)
      {
        if (!rotAval)
        {
          // either increment radixindex or freq
          if (digitalRead(pushPin) == LOW)
          {
            updatedisplay = 1;
            if (radix == 1000000)
              radix = 100000;
            else if (radix == 100000)
              radix = 10000;
            else if (radix == 10000)
              radix = 1000;
            else if (radix == 1000)
              radix = 100;
            else if (radix == 100)
              radix = 10;
            else if (radix == 10)
              radix = 1;
            else
              radix = 1000000;
          }
          else
          {
            freq = (freq + radix);
            if (freq > bandEnd)
              freq = bandEnd;
          }
          rotState = 2;                                   // CW 2
        }
      }
      else if (rotAval)
        rotState = 0;                                     // It was just a glitch on B, go back to start
      break;

    case 2:                                             // CW, wait for B high
      if (rotBval)
        rotState = 3;                                     // CW 3
      break;

    case 3:                                             // CW, wait for A high
      if (rotAval)
        rotState = 0;                                     // back to idle (detent) state
      break;

    case 11:                                             // CCW, wait for B low while A is low
      if (!rotAval)
      {
        if (!rotBval)
        {
          // either decrement radixindex or freq
          if (digitalRead(pushPin) == LOW)
          {
            updatedisplay = 1;
            if (radix == 1)
              radix = 10;
            else if (radix == 10)
              radix = 100;
            else if (radix == 100)
              radix = 1000;
            else if (radix == 1000)
              radix = 10000;
            else if (radix == 10000)
              radix = 100000;
            else if (radix == 100000)
              radix = 1000000;
            else
              radix = 1;
          }
          else
          {
            freq = (freq - radix);
            if (freq < bandStart)
              freq = bandStart;
          }
          rotState = 12;                                    // CCW 2
        }
      }
      else if (rotBval)
        rotState = 0;                                     // It was just a glitch on A, go back to start
      break;

    case 12:                                             // CCW, wait for A high
      if (rotAval)
        rotState = 13;                                      // CCW 3
      break;

    case 13:                                             // CCW, wait for B high
      if (rotBval)
        rotState = 0;                                     // back to idle (detent) state
      break;
  }
```

```
}


void UpdateDisplay()
{
  lcd.cursor();                                   // Turn on the cursor
  lcd.setCursor(0, 0);
  lcd.print("        ");
  lcd.setCursor(0, 0);
  lcd.print(freq);
  lcd.setCursor(10, 0);
  lcd.print("ZL2CTM");

  lcd.setCursor(0, 1);
  lcd.print("        ");
  lcd.setCursor(0, 1);

  if (freq > 9999999)
  {
    if (radix == 1)
      lcd.setCursor(7, 0);
    if (radix == 10)
      lcd.setCursor(6, 0);
    if (radix == 100)
      lcd.setCursor(5, 0);
    if (radix == 1000)
      lcd.setCursor(4, 0);
    if (radix == 10000)
      lcd.setCursor(3, 0);
    if (radix == 100000)
      lcd.setCursor(2, 0);
    if (radix == 1000000)
      lcd.setCursor(1, 0);

  }
  if (freq <= 9999999)
  {
    if (radix == 1)
      lcd.setCursor(6, 0);
    if (radix == 10)
      lcd.setCursor(5, 0);
    if (radix == 100)
      lcd.setCursor(4, 0);
    if (radix == 1000)
      lcd.setCursor(3, 0);
    if (radix == 10000)
      lcd.setCursor(2, 0);
    if (radix == 100000)
      lcd.setCursor(1, 0);
    if (radix == 1000000)
      lcd.setCursor(0, 0);
  }
}


void SendFrequency()
{
  si5351.set_freq((freq * 4) * 100ULL, SI5351_PLL_FIXED, SI5351_CLK0);

}
```

****

Transmit Test Configuration



Transmit Test Code

Note, the formatting has been messed up from cutting and pasting. Use auto format after pasting into the Arduino IDE)

```
// Libraries
#include <Wire.h>                        // I2C comms library
#include <si5351.h>                       // Si5351Jason library
#include <LiquidCrystal_I2C.h>            // LCD library
#include <Audio.h>                        // Teensy audio library

// Number of Filter Coefficients
#define NO_HILBERT_COEFFS 70              // Used to define the Hilbert transform filter arrays. More typical than 'const int'.

// Define Constants and Vaviables
static const long bandStart = 1000000;     // start of HF band
static const long bandEnd =   30000000;    // end of HF band
static const long bandInit =  3690000;     // where to initially set the frequency
//static const long bandInit =  14190000;  // where to initially set the frequency
volatile long oldfreq = 0;
volatile long freq = bandInit ;
volatile long radix = 1000;                // how much to change the frequency by clicking the rotary encoder will change this.
volatile int updatedisplay = 0;

static const int Mic_Gain = 0;            // Range is 0-63dB.
static const int Lineout_Gain = 20;        // Range is 13-31. 13 = 3.16 Vp-p, 31 = 1.16 Vp-p

// Rotary Encoder
static const int pushPin = 39;
static const int rotBPin = 36;
static const int rotAPin = 35;
volatile int rotState = 0;
volatile int rotAval = 1;
volatile int rotBval = 1;
volatile int rotAcc = 0;


// Iowa Hills Hilbert transform filter coefficients
const short Hilbert_Plus_45_Coeffs[NO_HILBERT_COEFFS] = {
  (short)(32768 * -0.000287988910943357),
  (short)(32768 * -0.000383511439791303),
  (short)(32768 * -0.000468041804899774),
  (short)(32768 * -0.000529324432676899),
  (short)(32768 * -0.000569479602046985),
  (short)(32768 * -0.000616670267768531),
  (short)(32768 * -0.000731530748681977),
  (short)(32768 * -0.001002372095321225),
  (short)(32768 * -0.001525299390682192),
  (short)(32768 * -0.002370114347025230),
  (short)(32768 * -0.003539247773172147),
  (short)(32768 * -0.004932965382552984),
  (short)(32768 * -0.006337182914262393),
  (short)(32768 * -0.007448193692118567),
  (short)(32768 * -0.007940501940620482),
  (short)(32768 * -0.007570802072162988),
  (short)(32768 * -0.006296120449841751),
  (short)(32768 * -0.004371955618154949),
  (short)(32768 * -0.002391875073164555),
  (short)(32768 * -0.001236984700413469),
  (short)(32768 * -0.001922560128827416),
  (short)(32768 * -0.005356720327533458),
  (short)(32768 * -0.012055656297010635),
  (short)(32768 * -0.021882952959947619),
  (short)(32768 * -0.033888748300090733),
  (short)(32768 * -0.046312736456333638),
  (short)(32768 * -0.056783367797647665),
  (short)(32768 * -0.062699937453677912),
  (short)(32768 * -0.061735375084135742),
  (short)(32768 * -0.052358513976237808),
  (short)(32768 * -0.034257179158167443),
  (short)(32768 * -0.008554500746482946),
  (short)(32768 * 0.022249911747384360),
  (short)(32768 * 0.054622962942346594),
  (short)(32768 * 0.084568844473140448),
  (short)(32768 * 0.108316122839950818),
  (short)(32768 * 0.122979341462627859),
  (short)(32768 * 0.127056096658453188),
  (short)(32768 * 0.120656295327679283),
  (short)(32768 * 0.105420364259485699),
  (short)(32768 * 0.084152608145489444),
  (short)(32768 * 0.060257510644444748),
  (short)(32768 * 0.037105711921879434),
  (short)(32768 * 0.017464092086704748),
  (short)(32768 * 0.003100559033325746),
  (short)(32768 * -0.005373489802481697),
  (short)(32768 * -0.008418211280310166),
  (short)(32768 * -0.007286730644726664),
  (short)(32768 * -0.003638388931163832),
  (short)(32768 * 0.000858330713630433),
  (short)(32768 * 0.004847436504682235),
  (short)(32768 * 0.007476399317750315),
  (short)(32768 * 0.008440227567663121),
  (short)(32768 * 0.007898970420636600),
  (short)(32768 * 0.006314366257036837),
  (short)(32768 * 0.004261033495040515),
  (short)(32768 * 0.002261843500794377),
  (short)(32768 * 0.000680212977485724),
  (short)(32768 * -0.000319493110301691),
  (short)(32768 * -0.000751893569425181),
  (short)(32768 * -0.000752248417868501),
  (short)(32768 * -0.000505487955986662),
  (short)(32768 * -0.000184645628631330),
  (short)(32768 * 0.000087913008490067),
  (short)(32768 * 0.000253106348867209),
  (short)(32768 * 0.000306473486382603),
  (short)(32768 * 0.000277637042003169),
  (short)(32768 * 0.000207782317481292),
  (short)(32768 * 0.000132446796990356),
  (short)(32768 * 0.000072894261560354)
};

// Iowa Hills Hilbert transform filter coefficients
const short Hilbert_Minus_45_Coeffs[NO_HILBERT_COEFFS] = {
  (short)(32768 * -0.000072894261560345),
  (short)(32768 * -0.000132446796990344),
  (short)(32768 * -0.000207782317481281),
  (short)(32768 * -0.000277637042003168),
  (short)(32768 * -0.000306473486382623),
  (short)(32768 * -0.000253106348867259),
  (short)(32768 * -0.000087913008490148),
  (short)(32768 * 0.000184645628631233),
  (short)(32768 * 0.000505487955986583),
  (short)(32768 * 0.000752248417868491),
  (short)(32768 * 0.000751893569425298),
  (short)(32768 * 0.000319493110301983),
  (short)(32768 * -0.000680212977485245),
  (short)(32768 * -0.002261843500793748),
  (short)(32768 * -0.004261033495039842),
  (short)(32768 * -0.006314366257036280),
  (short)(32768 * -0.007898970420636345),
  (short)(32768 * -0.008440227567663343),
  (short)(32768 * -0.007476399317751102),
  (short)(32768 * -0.004847436504683540),
  (short)(32768 * -0.000858330713632029),
  (short)(32768 * 0.003638388931162351),
  (short)(32768 * 0.007286730644725833),
  (short)(32768 * 0.008418211280310565),
  (short)(32768 * 0.005373489802483816),
  (short)(32768 * -0.003100559033321630),
  (short)(32768 * -0.017464092086698697),
  (short)(32768 * -0.037105711921871905),
  (short)(32768 * -0.060257510644436532),
  (short)(32768 * -0.084152608145481672),
  (short)(32768 * -0.105420364259479538),
  (short)(32768 * -0.120656295327675800),
  (short)(32768 * -0.127056096658453216),
  (short)(32768 * -0.122979341462631633),
  (short)(32768 * -0.108316122839958146),
  (short)(32768 * -0.084568844473150454),
  (short)(32768 * -0.054622962942358168),
  (short)(32768 * -0.022249911747396132),
  (short)(32768 * 0.008554500746472333),
  (short)(32768 * 0.034257179158159054),
  (short)(32768 * 0.052358513976232306),
  (short)(32768 * 0.061735375084133286),
  (short)(32768 * 0.062699937453678217),
  (short)(32768 * 0.056783367797650072),
  (short)(32768 * 0.046312736456337288),
  (short)(32768 * 0.033888748300094730),
  (short)(32768 * 0.021882952959951244),
  (short)(32768 * 0.012055656297013388),
  (short)(32768 * 0.005356720327535105),
  (short)(32768 * 0.001922560128828006),
  (short)(32768 * 0.001236984700413229),
  (short)(32768 * 0.002391875073163812),
  (short)(32768 * 0.004371955618154038),
  (short)(32768 * 0.006296120449840938),
  (short)(32768 * 0.007570802072162439),
  (short)(32768 * 0.007940501940620253),
  (short)(32768 * 0.007448193692118624),
  (short)(32768 * 0.006337182914262643),
  (short)(32768 * 0.004932965382553323),
  (short)(32768 * 0.003539247773172483),
  (short)(32768 * 0.002370114347025498),
  (short)(32768 * 0.001525299390682370),
  (short)(32768 * 0.001002372095321316),
  (short)(32768 * 0.000731530748682004),
  (short)(32768 * 0.000616670267768521),
  (short)(32768 * 0.000569479602046963),
  (short)(32768 * 0.000529324432676881),
  (short)(32768 * 0.000468041804899765),
  (short)(32768 * 0.000383511439791304),
  (short)(32768 * 0.000287988910943362)
};


// Instantiate the Objects
LiquidCrystal_I2C lcd(0x3F, 16, 2);       // Name for the LCD. Set the LCD address to either 0x27
or 0x3F for a 16 chars and 2 line display
Si5351 si5351;                             // Name for the Si5351 DDS
AudioControlSGTL5000    audioShield;      // Name for the Teensy audio CODEC on the audio
shield
```

```
// Audio shield
AudioInputI2S           audioInput;                                          // Name for the input to the audio
shield (either line-in or mic)
AudioOutputI2S          audioOutput;                                         // Name for the output of the audio
shield (either headphones or line-out)
// Transmitter
AudioFilterFIR          TX_Hilbert_Plus_45;                                  // Name for the TX +45 Hilbert
transform
AudioFilterFIR          TX_Hilbert_Minus_45;                                 // Name for the TX +45 Hilbert
transform
AudioMixer4             TX_I_Sideband_Switch;                                // Name for the sideband
switching summer for the I channel

// Audio connections
AudioConnection         patchCord50(audioInput, 0, TX_Hilbert_Plus_45, 0);              // Mic audio
to Hilbert transform +45
AudioConnection         patchCord55(audioInput, 0, TX_Hilbert_Minus_45, 0);             // Mic audio
to  Hilbert transform -45
AudioConnection         patchCord60(TX_Hilbert_Plus_45, 0, TX_I_Sideband_Switch, 0);    //
Hilbert transform +45 to receiver summer
AudioConnection         patchCord65(TX_I_Sideband_Switch, 0, audioOutput, 0);           // Output
to the NE612
AudioConnection         patchCord70(TX_Hilbert_Minus_45, 0, audioOutput, 1);            // Output to
the NE612


void setup()
{
  // Setup input switches
  pinMode(rotAPin, INPUT);
  pinMode(rotBPin, INPUT);
  pinMode(pushPin, INPUT);
  digitalWrite(rotAPin, HIGH);
  digitalWrite(rotBPin, HIGH);
  digitalWrite(pushPin, HIGH);

  // Setup interrupt pins
  attachInterrupt(digitalPinToInterrupt(rotAPin), ISRrotAChange, CHANGE);
  attachInterrupt(digitalPinToInterrupt(rotBPin), ISRrotBChange, CHANGE);

  // Setup the lcd
  lcd.begin();
  lcd.backlight();

  // Setup the DDS
  si5351.init(SI5351_CRYSTAL_LOAD_8PF, 0);
  si5351.set_pll(SI5351_PLL_FIXED, SI5351_PLLA);
  si5351.drive_strength(SI5351_CLK0, SI5351_DRIVE_8MA);
  si5351.set_freq((freq * 100ULL), SI5351_PLL_FIXED, SI5351_CLK0);

  // Setup the audio shield
  AudioNoInterrupts();
  AudioMemory(16);
  audioShield.enable();
  AudioInterrupts();

  // Setup transceiver mode
  Turn_On_Transmitter();
  UpdateDisplay();
}

void loop()
{
  if (freq != oldfreq)                // Check to see if the frequency has changed. If so, update
everything.
  {
    UpdateDisplay();
    SendFrequency();
    oldfreq = freq;
  }

  if (digitalRead(pushPin) == LOW)    // Update cursor, but also stop it from flickering
  {
    delay(10);
    while (digitalRead(pushPin) == LOW)
    {
      if (updatedisplay == 1)
      {
        UpdateDisplay();
        updatedisplay = 0;
      }
    }
    delay(50);
  }
}


void Turn_On_Transmitter()
{
  AudioNoInterrupts();
  audioShield.inputSelect(AUDIO_INPUT_MIC);
  audioShield.micGain(Mic_Gain);
  audioShield.unmuteLineout();                                        // Output to the NE612s
  audioShield.lineOutLevel(Lineout_Gain);
  TX_Hilbert_Plus_45.begin(Hilbert_Plus_45_Coeffs, NO_HILBERT_COEFFS);
  TX_Hilbert_Minus_45.begin(Hilbert_Minus_45_Coeffs, NO_HILBERT_COEFFS);

  if (freq <= 9999999)          // LSB
  {
    TX_I_Sideband_Switch.gain(0, 1);
  }
  if (freq > 9999999)           // USB
  {
    TX_I_Sideband_Switch.gain(0, -1);
  }

  AudioInterrupts();
}


// Interrupt routines
void ISRrotAChange()
{
  if (digitalRead(rotAPin))
  {
    rotAval = 1;
    UpdateRot();
  }
  else
  {
    rotAval = 0;
    UpdateRot();
  }
}


void ISRrotBChange()
{
  if (digitalRead(rotBPin))
  {
    rotBval = 1;
    UpdateRot();
  }
  else
  {
    rotBval = 0;
    UpdateRot();
  }
}


void UpdateRot()
{
  switch (rotState)
  {

    case 0:                                          // Idle state, look for direction
      if (!rotBval)
        rotState = 1;                                // CW 1
      if (!rotAval)
        rotState = 11;                               // CCW 1
      break;

    case 1:                                          // CW, wait for A low while B is low
      if (!rotBval)
      {
        if (!rotAval)
    {
      // either increment radixindex or freq
      if (digitalRead(pushPin) == LOW)
      {
        updatedisplay = 1;
        if (radix == 1000000)
          radix = 100000;
        else if (radix == 100000)
          radix = 10000;
        else if (radix == 10000)
          radix = 1000;
        else if (radix == 1000)
          radix = 100;
        else if (radix == 100)
          radix = 10;
        else if (radix == 10)
          radix = 1;
        else
          radix = 1000000;
      }
      else
      {
        freq = (freq + radix);
        if (freq > bandEnd)
          freq = bandEnd;
      }
      rotState = 2;                              // CW 2
    }
  }
  else if (rotAval)
    rotState = 0;                                // It was just a glitch on B, go back to start
  break;

case 2:                                        // CW, wait for B high
  if (rotBval)
    rotState = 3;                                // CW 3
  break;

case 3:                                        // CW, wait for A high
  if (rotAval)
    rotState = 0;                                // back to idle (detent) state
  break;

case 11:                                        // CCW, wait for B low while A is low
  if (!rotAval)
  {
    if (!rotBval)
    {
      // either decrement radixindex or freq
      if (digitalRead(pushPin) == LOW)
      {
        updatedisplay = 1;
        if (radix == 1)
          radix = 10;
        else if (radix == 10)
          radix = 100;
        else if (radix == 100)
          radix = 1000;
        else if (radix == 1000)
          radix = 10000;
        else if (radix == 10000)
          radix = 100000;
        else if (radix == 100000)
          radix = 1000000;
        else
          radix = 1;
      }
      else
      {
        freq = (freq - radix);
        if (freq < bandStart)
          freq = bandStart;
      }
      rotState = 12;                              // CCW 2
    }
  }
  else if (rotBval)
    rotState = 0;                                // It was just a glitch on A, go back to start
  break;
```

```
    case 12:                          // CCW, wait for A high
      if (rotAval)
        rotState = 13;                  // CCW 3
      break;

    case 13:                          // CCW, wait for B high
      if (rotBval)
        rotState = 0;                   // back to idle (detent) state
      break;
  }
}


void UpdateDisplay()
{
  lcd.cursor();                       // Turn on the cursor
  lcd.setCursor(0, 0);
  lcd.print("        ");
  lcd.setCursor(0, 0);
  lcd.print(freq);
  lcd.setCursor(10, 0);
  lcd.print("ZL2CTM");

  lcd.setCursor(0, 1);
  lcd.print("        ");
  lcd.setCursor(0, 1);

  if (freq > 9999999)
  {
    if (radix == 1)
      lcd.setCursor(7, 0);
    if (radix == 10)
      lcd.setCursor(6, 0);
    if (radix == 100)
      lcd.setCursor(5, 0);
    if (radix == 1000)
      lcd.setCursor(4, 0);
    if (radix == 10000)
      lcd.setCursor(3, 0);
    if (radix == 100000)
      lcd.setCursor(2, 0);
    if (radix == 1000000)
      lcd.setCursor(1, 0);

  }
  if (freq <= 9999999)
  {
    if (radix == 1)
      lcd.setCursor(6, 0);
    if (radix == 10)
      lcd.setCursor(5, 0);
    if (radix == 100)
      lcd.setCursor(4, 0);
    if (radix == 1000)
      lcd.setCursor(3, 0);
    if (radix == 10000)
      lcd.setCursor(2, 0);
    if (radix == 100000)
      lcd.setCursor(1, 0);
    if (radix == 1000000)
      lcd.setCursor(0, 0);
  }
}


void SendFrequency()
{
  si5351.set_freq((freq * 4) * 100ULL, SI5351_PLL_FIXED, SI5351_CLK0);

}
```

****

Posted by Charlie Morris at 22:42

Labels: Hilbert Transform, SDR, Teensy, ZL2CTM

9 comments:

Andrey Begunov 21 March 2018 at 13:20

Charlie, thanks a lot for the great video and explanation of the SDR theory and practice, looking forward for future video! 73! UT9UF

Reply

Charlie Morris 21 March 2018 at 19:27

Cheers Andrey. My intent is to explain things in the videos and not here. My typing is waaaaay too slow!

73s
Charlie ZL2CTM

Reply

Jason 28 March 2018 at 18:50

Great job with the videos as always! I wanted to ask if you've done any segments walking through how you peak the BPF? If not, I think that would be a good addition. I know you spoke on the theory of it in the last video, but seeing thebprocthe is very helpful!

Thx
W4UNX

Reply

Replies

Charlie Morris 28 March 2018 at 19:08

Thanks Jason. I might have tuned a BPF a while back, but I cannot recall which video. I'll certainly look to do it again.
73s
Charlie ZL2CTM

Reply

Bob 29 March 2018 at 17:03

Really enjoying this series Charlie. What is the source of the Liquid_crystal_I2C library?

Reply

Replies

Charlie Morris 30 March 2018 at 15:17

Thanks Bob. As for the library, I'm prety sure I got it from here:

https://github.com/marcoschwartz/LiquidCrystal_I2C

Charlie

Bob 8 April 2018 at 10:17

This comment has been removed by the author.

Bob 30 May 2018 at 10:52

Charlie I have a working SSB SDR reciever!! Thx for blazing the trail with this series.

Reply

Dave Metzler 3 September 2018 at 14:51

Have you posted the complete Teensy SDR transceiver code anywhere?

Reply

Enter your comment...

Comment as: Google Account

Publish **Preview**

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)