```
//20180808
const int LCDEnable = 10;  //LCD enable
const int SCLK = 11;   //I2C clocking pin 11
const int SDAT = 12;  //I2C data pin 12
const int Pbswitch = 4; //Switch pin 4
const int AttLatch1 = 5;  //Attenuator 1 latch pin 8
const int AttLatch2 = 6;  //Attenuator 2 latch pin 6
const int AF0 = 7; //Filter control A0
const int AF1 = 8; //Filter control A1
const int AF2 = 9; //Filter control A2
int Band; //Frequency band
int analogPin = 5;  //Analog input pin
byte PushButton = LOW;  //Initial state of pushbutton
int OptoA = 2; //Optalencoder phase A pin 2
int OptoB = 3; //Optalencoder phase B pin 3
int PhaseA; //Opticalencoder phase A
int PhaseB; //Opticalencoder phase B
int analogVal = 512;  //Set initial analog value to mid scale
int N1; //Si570 N1 divider
int HsDiv;  //Si570 Hs_Div divider
unsigned long Fout = 10000000;                    //Frequency to be output by Si570 (Hz)
unsigned long Fdisplay = Fout; //Frequency on the display
int Rdisplay = 0; //RF on the display
int RdisplayMax = 70; //Initial rf level max in minus dBm
int EAttDisplay = 0;  //External attenuator value on the display
int EAttDisplayMax = 60;  //External attenuator maximum in dB
float PAdisplay = 0; //Power adjustment in dB (-0.75 to +0.75 in 0.25 steps)
float Finc = .00001;  //Frequency increment for the sweep function
float Fdco; //Si570 Digitally Controlled Oscillator
float Fxtal = 114.29214;                    //Si570 crystal frequency (MHz). This can adjust
frequency. Increase => decrease in output frequency.
int IRdisplay;  //Float of RF on display
int RFadj; //Total attenuation needed to get 0 dBM including all insertion losses
float FRFadj;  //Float of total attenuation
int ATTvalu1; //Attenuation value to attenuator 1
int ATTvalu2; //Attenuation value to attenuators 2 ,3, and 4
const int FdcoLower = 4850; //Si570 DCO lower frequency limit (MHz)
const int FdcoUpper = 5670; //Si570 DCO upper limit (MHz)
const int debounce = 30;  //Debounce time in milliseconds
byte SiRegOut[6]; //Si570 registers array to be put out
byte SiRegIn[6];  //Si570 registers array read in
int FreqDig = 9; //Initialize frequency digit to LCD column 7 (MS digit)
int SpeedDig = 11;  //Initialize SWEEP speed digit to LCD column 11 (MS digit)
byte BlankFlag;  //Used to blank the SWEEP field in the sweep mode
byte ModeSwitch = LOW;  //Set mode switch to LOW => frequency change. HIGH => mode change
byte DigSwitch = LOW;  //Digital switch initial value
unsigned long time1; //Start time for timing purposes
unsigned long time2; //Stop time for timing purposes
```

```
    unsigned long time3; //Start time for timing purposes
    unsigned long time4; //Stop time for timing purposes
    unsigned long SwitchStartTime;  //Switch transition start time
    unsigned long SwitchStopTime;  //Switch transition stop time
    int prevSwitch = HIGH; //Initial switch position
    int Transition = LOW; //No switch transition initially
    volatile int ChangeA = LOW; //initialization of Opto interrupt change A variable
    volatile int ChangeB = LOW; //initialization of Opto interrupt changeB variable
    int OptoChange; //Increment or Decrement from CW or CCW rotation of the optical encoder
    int ChangeFlag = HIGH;  //Indicates the LCD needs changing from a switch change
    int NoChangeFlag = LOW; //Indicates no change in the LCD from this switch change
    int ParSelCol = 0; //Initial parameter column position
    int ParSelRow = 1; //Initial parameter row position
    int ParNum = 1; //Initialize parameter number

    //*********************************************************************************
***************

    void setup() {
    pinMode(SCLK,OUTPUT);   //Set the I2C clock pin for output
    pinMode(SDAT,OUTPUT);   //Set the I2C data pin for output
    pinMode(OptoA, INPUT);  //Set the optical decoder phase A for input
    pinMode(OptoB, INPUT);  //Set the optical decoder phase B for input
    pinMode(Pbswitch, INPUT_PULLUP);  //Set the push button for input with a pull up resistor
    pinMode(AF0, OUTPUT);  //Filter decoder A0
    pinMode(AF1, OUTPUT);  //Filter decoder A1
    pinMode(AF2, OUTPUT);  //Filter decoder A2
    pinMode(A4, OUTPUT);  //Filter encoder enable line
    pinMode(A3, OUTPUT);  //30-dB Attenuator switch
    digitalWrite(AF0, LOW);  //Initialize filter control
    digitalWrite(AF1, LOW);  //Initialize filter control
    digitalWrite(AF2, LOW);  //Initialize filter control
    pinMode(AttLatch1, OUTPUT); //Attenuator Latch #1
    pinMode(AttLatch2, OUTPUT); //Attenuator Latch #2
    digitalWrite(AttLatch1, LOW); //Initialize
    digitalWrite(SCLK, LOW);//Initialize
    attachInterrupt(digitalPinToInterrupt(2),Opto_A,CHANGE);
    attachInterrupt(digitalPinToInterrupt(3),Opto_B,CHANGE);
    interrupts(); //Turn on interrupts
    Serial.begin(9600); //Set up the monitor window port
    delay(1); //Delay for LCD setup
    InitializeLCD(); // Set up LCD
    LCDHeaders(); //Write LCD Headers
    FrequencyCheck(); //Do a frequency check for valid frequency
    FilterControl();  //Switch the filters for different frequencies
    digitalWrite(A4, HIGH); //Enable filter decoder
    digitalWrite(A3, HIGH); //30-db Attenuator out initially
    SetContrast(50);  //Set LCD contrast
```

```
    }

    //*********************************************************************************
***************

    void loop()  {
    ChangeLCD();  //Update the LCD
    SwitchRead(); //Read the switches
    OpticalEncoder(); //Increment parameters with the optical encoder
    DigitSelect();  //Select the frequency digit to be changed
    RFCorrection(); //Compute RF level correction and set the attenuator
    //delay(100);
    }

    //*********************************************************************************
***************

    //OpticalEncoder decodes the rotation of the encoder and determines the rotation direction
    void OpticalEncoder()  {
    if(ChangeA == HIGH) {
     ChangeFlag = HIGH;
     NoChangeFlag = HIGH;
     ChangeA = LOW;  //Reset interrupt flag
     PhaseA = digitalRead(OptoA);  //Get optical encoder phase A state
     PhaseB = digitalRead(OptoB);  //Get optical encoder phase B state
      if(((PhaseA == HIGH) && (PhaseB == LOW)) || ((PhaseA == LOW) && (PhaseB == HIGH)))  {
       if(ParNum == 1) {
         FrequencyDigitIncrement();
       }
       else if(ParNum == 2)  {
         Rdisplay = Rdisplay - 1;  //Increment RF output
         if(Rdisplay < 0) {Rdisplay = 0; }
       }
       else if(ParNum == 3)  {
         EAttDisplay = EAttDisplay + 10;
         if(EAttDisplay > EAttDisplayMax) {EAttDisplay = EAttDisplayMax;}
       }
      }
      else if(((PhaseA == HIGH) && (PhaseB == HIGH)) || ((PhaseA == LOW) && (PhaseB == LOW)))  {
       if(ParNum == 1) {
         FrequencyDigitDecrement();
       }
       else if(ParNum == 2)  {
         Rdisplay = Rdisplay + 1;  //Decrement RF output from a less negative number
         if(Rdisplay > RdisplayMax) {Rdisplay = RdisplayMax; }
       }
       else if(ParNum == 3)  {
         EAttDisplay = EAttDisplay - 10;
```

```
    if(EAttDisplay < 0) {EAttDisplay = 0;}
   }
  }
}
else if(ChangeB == HIGH) {
  ChangeFlag = HIGH;
  NoChangeFlag = HIGH;
  ChangeB = LOW;  //Reset interrupt flag
  PhaseA = digitalRead(OptoA);  //Get optical encoder phase A state
  PhaseB = digitalRead(OptoB);  //Get optical encoder phase B state
  if(((PhaseB == HIGH) && (PhaseA == HIGH)) || ((PhaseB == LOW) && (PhaseA == LOW)))  {
   if(ParNum == 1) {
    FrequencyDigitIncrement();
   }
   else if(ParNum == 2)  {
    Rdisplay = Rdisplay - 1;  //Increment RF output from a less negative number
    if(Rdisplay < 0) {Rdisplay = 0; }
   }
   else if(ParNum == 3)  {
    EAttDisplay = EAttDisplay + 10;
    if(EAttDisplay > EAttDisplayMax) {EAttDisplay = EAttDisplayMax;}
   }
  }
  else if(((PhaseB == HIGH) && (PhaseA == LOW)) || ((PhaseB == LOW) && (PhaseA == HIGH)))  {
   if(ParNum == 1) {
    FrequencyDigitDecrement();
   }
   else if(ParNum == 2)  {
    Rdisplay = Rdisplay + 1;  //Decrement RF output from a less negative number
    if(Rdisplay > RdisplayMax) {Rdisplay = RdisplayMax; }
   }
   else if(ParNum == 3)  {
    EAttDisplay = EAttDisplay - 10;
    if(EAttDisplay < 0) {EAttDisplay = 0;}
   }
  }

}
}

//Opto_A is an interrupt routine to detect a change in opticalencoder phase A
void Opto_A() {
ChangeA = HIGH;
}

//Opto_B is an interrupt routine to detect a change in opticalencoder phase B
void Opto_B() {
ChangeB = HIGH;
```

```
}

void FrequencyDigitIncrement() {
   if(ModeSwitch == LOW)  {
     //Here we increment the correct frequency digit
     switch(FreqDig) {
       case 7:
         Fdisplay = Fdisplay + 100000000;
         break;
       case 8:
         Fdisplay = Fdisplay + 10000000;
         break;
       case 9:
         Fdisplay = Fdisplay + 1000000;
         break;
       case 11:
         Fdisplay = Fdisplay + 100000;
         break;
       case 12:
         Fdisplay = Fdisplay + 10000;
         break;
       case 13:
         Fdisplay = Fdisplay + 1000;
         break;
       case 14:
         Fdisplay = Fdisplay + 100;
         break;
       case 15:
         Fdisplay = Fdisplay + 10;
         break;
     }
   }
FrequencyCheck();
FilterControl();
}


void FrequencyDigitDecrement() {
   if(ModeSwitch == LOW)  {
     //Here we decrement the correct frequency digit
     switch(FreqDig) {
       case 7:
         if (Fdisplay != 500000)
         {
           Fdisplay = Fdisplay - 100000000;
         }
         break;
       case 8:
```

```
        if (Fdisplay != 500000)
        {
          Fdisplay = Fdisplay - 10000000;
        }
         break;
       case 9:
        if (Fdisplay != 500000)
        {
          Fdisplay = Fdisplay - 1000000;
        }
         break;
       case 11:
        Fdisplay = Fdisplay - 100000;
        break;
       case 12:
        Fdisplay = Fdisplay - 10000;
        break;
       case 13:
        Fdisplay = Fdisplay - 1000;
        break;
       case 14:
        Fdisplay = Fdisplay - 100;
        break;
       case 15:
        Fdisplay = Fdisplay - 10;
        break;
     }
   }
FrequencyCheck();
}

//FrequencyCheck makes sure the frequency is correct and sets it into the SI570
void FrequencyCheck() {
 if(Fdisplay >= 128000000) { Fdisplay = 128000000; }  //Set upper frequency limit
 if(Fdisplay <= 500000) { Fdisplay = 500000; }  //Set lower frequency limit
 if(Fdisplay < 4000000)  { Fout = 8 * Fdisplay;  }
 else  { Fout = Fdisplay;  }
 SetFreq();
}


//FilterControl switches in the proper filter depending on frequency
void FilterControl()  {
if((Fdisplay >= 500000) &  (Fdisplay < 1000000)) {
 //filter 5
 Band = 1;
 digitalWrite(AF2, HIGH);
 digitalWrite(AF1, LOW);
```

```
  digitalWrite(AF0, HIGH);
 }
 else if((Fdisplay >= 1000000) &  (Fdisplay < 2000000))  {
  //filter 6
  Band = 2;
  digitalWrite(AF2, HIGH);
  digitalWrite(AF1, HIGH);
  digitalWrite(AF0, LOW);
 }
 else if((Fdisplay >= 2000000) &  (Fdisplay < 4000000))  {
  //filter 7
  Band = 3;
  digitalWrite(AF2, HIGH);
  digitalWrite(AF1, HIGH);
  digitalWrite(AF0, HIGH);
 }
 else if((Fdisplay >= 4000000) &  (Fdisplay < 8000000))  {
  //filter 0
  Band = 4;
  digitalWrite(AF2, LOW);
  digitalWrite(AF1, LOW);
  digitalWrite(AF0, LOW);
 }
 else if((Fdisplay >= 8000000) &  (Fdisplay < 12800000))  {
  //filter 1
  Band = 5;
  digitalWrite(AF2, LOW);
  digitalWrite(AF1, LOW);
  digitalWrite(AF0, HIGH);
 }
 if((Fdisplay >= 12800000) &  (Fdisplay < 32000000))  {
  //filter 2
  Band = 6;
  digitalWrite(AF2, LOW);
  digitalWrite(AF1, HIGH);
  digitalWrite(AF0, LOW);
 }
 else if((Fdisplay >= 32000000) &  (Fdisplay < 64000000))  {
  //filter 3
  Band = 7;
  digitalWrite(AF2, LOW);
  digitalWrite(AF1, HIGH);
  digitalWrite(AF0, HIGH);
 }
 else if((Fdisplay >= 64000000) &  (Fdisplay <= 128000000))  {
  //filter 4
  Band = 8;
  digitalWrite(AF2, HIGH);
```

```
  digitalWrite(AF1, LOW);
  digitalWrite(AF0, LOW);
 }
 }

 //SwitchRead reads the push button switch on the opticalencoder, debounces it, and signals a good transition
 void SwitchRead()  {
  PushButton = digitalRead(Pbswitch);
  if(PushButton != prevSwitch)  {
   delay(debounce);  //Delay for bouncing
   PushButton = digitalRead(Pbswitch); //Check switch again
   if(PushButton != prevSwitch) {
    if(PushButton == LOW) {
     prevSwitch = LOW; //Set next previous switch state
    }
    else if(PushButton == HIGH) {
     prevSwitch = HIGH;  //Set next previous switch state
     SelectPar();  //The pushbutton has been pushed, so select next parameter
     ParNum = ParNum + 1;  //Increment parameter number
     if(ParNum > 3) {ParNum = 1; }
     ChangeFlag = HIGH;
     SelectPar();
   }
  }
 }
 }

 //DigitSelect() uses an analog input to select digits either right or left
 void DigitSelect()  {
  if(DigSwitch == LOW) {
   analogVal = analogRead(analogPin);  //Read the analog channel
   if((analogVal < 256) || (analogVal > 768))  {
    delay(debounce);  //Delay for switch bounce
    analogVal = analogRead(analogPin);  //Read the analog channel again
    if((analogVal) < 256)  {
     ChangeFlag = HIGH;
     NoChangeFlag = HIGH;
     FreqDig = FreqDig - 1;  //Move selected digit indicator to the left
     if(FreqDig == 10) { FreqDig = 9;  } //Skip over the decimal point
     if(FreqDig < 7) { FreqDig = 7; }  //Stop at left end point
     DigSwitch = HIGH; //Reset the button push
    }
    else if(analogVal > 768)  {
     ChangeFlag = HIGH;
     NoChangeFlag = HIGH;
     FreqDig = FreqDig +1; //Move selected digit indicator to the right
     if(FreqDig == 10) { FreqDig = 11;  }  //Skip over the decimal point
```

```
      if(FreqDig > 15) { FreqDig = 15; }  //Stop at right end point
      DigSwitch = HIGH;
     }
    }
  }
  analogVal = analogRead(analogPin);
   if((analogVal > 256) && (analogVal < 768))  {
    DigSwitch = LOW;
   }
}



//LCDHeaders sets up the LCD headers
void LCDHeaders(){
 CursorHome();
 LCDWriteString("W0IVJ SI570 Sig Gen");
 SetCursor(1 ,1); //Set cursor for frequency display
 LCDWriteString("FREQ: "); //Display frequency label
 SetCursor(1, 2); //Set cursor for attenuation display
 LCDWriteString("RF PWR (dBM): 0");  //Display attenuation label
 SetCursor(1 ,3); //Set cursor for PWR adjustment
 LCDWriteString("External Atten: 0");
 SetCursor(ParSelCol ,ParSelRow); //Set cursor frequency change
 LCDWriteString(">");
}

//ChangeLCD writes the LCD with current changes
void ChangeLCD()  {
 if(ParNum == 1) {
  if(ChangeFlag == HIGH) {
   int x;
   String SFdisplay;
   String PrintString;
   String StartString;
   String EndString;
   SetCursor(7 ,1);
    SFdisplay = String(Fdisplay);
    if((Fdisplay >= 500000) && (Fdisplay <= 999999))  {
     x = SFdisplay.length();
     x = x - 1;
     EndString = SFdisplay.substring(0, x);
     PrintString = "000." + EndString;
    }
    else if((Fdisplay >= 1000000) && (Fdisplay <= 9999999))  {
     x = SFdisplay.length();
     x = x - 1;
     StartString = SFdisplay.substring(0, 1);
```

```
      EndString = SFdisplay.substring(1, x);
      PrintString = "00" + StartString + "." + EndString;
     }
     else if((Fdisplay >= 10000000) && (Fdisplay <= 99999999)) {
      x = SFdisplay.length();
      x = x - 1;
      StartString = SFdisplay.substring(0, 2);
      EndString = SFdisplay.substring(2, x);
      PrintString = "0" + StartString + "." + EndString;
     }
     else if((Fdisplay >= 100000000) && (Fdisplay <= 128000000)) {
      x = SFdisplay.length();
      x = x - 1;
      StartString = SFdisplay.substring(0, 3);
      EndString = SFdisplay.substring(3, x);
      PrintString = StartString + "." + EndString;
     }
     LCDWriteString(PrintString);
     SetCursor(FreqDig, 1);
     UnderlineOn();
     ChangeFlag = LOW;
     FilterControl();
  }
 }
 else if(ParNum == 2)  {
  if(ChangeFlag == HIGH)  {
    String SRdisplay;
    if(Rdisplay == 0) { SRdisplay = String(Rdisplay); }
    else if(Rdisplay > 0) {SRdisplay = ("-") + (String(Rdisplay)); }
    SetCursor(15, 2);
    LCDWriteString("    ");
    int Radisplay = SRdisplay.toInt();
    Radisplay = Radisplay - EAttDisplay;
    SRdisplay = String(Radisplay);
    SetCursor(15, 2);
    LCDWriteString(SRdisplay);
    ChangeFlag = LOW;
  }
 }
 else if(ParNum == 3)
 {
  if(ChangeFlag == HIGH)
  {
    //Display of attenuation is repeated here for External Attenuation
    String SRdisplay;
    if(Rdisplay == 0) { SRdisplay = String(Rdisplay); }
    else if(Rdisplay > 0) {SRdisplay = ("-") + (String(Rdisplay)); }
    SetCursor(15, 2);
```

```
    LCDWriteString("    ");
    int Radisplay = SRdisplay.toInt();
    Radisplay = Radisplay - EAttDisplay;
    SRdisplay = String(Radisplay);
    SetCursor(15, 2);
    LCDWriteString(SRdisplay);
    SetCursor(17, 3);
    LCDWriteString("  ");
    SetCursor(17, 3);
    LCDWriteString(String(EAttDisplay));
    ChangeFlag = LOW;
   }
  }

}

//Sweeper sweeps the frequency between two frequencies up and down at a specified step and rate
void Sweeper(float StartFreq, float StopFreq, float FreqIncr, int SweepSpeed) {
for(Fout = StartFreq; Fout <= StopFreq; Fout = Fout + FreqIncr) {
ChangeFreq();
delay(SweepSpeed);
  }
for(Fout = StopFreq; Fout >= StartFreq; Fout = Fout - FreqIncr) {
ChangeFreq();
delay(SweepSpeed);
  }
}

//send a START to SI570
 void StartTransfer(){
  digitalWrite(SCLK,HIGH);
  digitalWrite(SDAT,HIGH);
  digitalWrite(SDAT,LOW);
  digitalWrite(SCLK,LOW);

}

void DividerCalc() {
  const int FdcoAverage = 5260;
  int N1Odd;
  float FFout = Fout;
  FFout = FFout / 1000000;
//  FFout = 126.123456;
  N1 = FdcoAverage / (FFout * HsDiv);   //Calculate an N1 value
  N1Odd = N1 % 2;   //Correct N1 if it is odd
  if(N1Odd == 1)  {
     if (N1 !=1)  {
      N1 = N1 +1;
```

```
    }
  }
  Fdco = FFout * HsDiv * N1;   //Calculate the real Fdco

}

//Calculate Si570 registers
void RegCalc() {
  int N1Reg;    //Si570 N1 register
  int HsDivReg; //Si570 Hs_Div register
  byte TempReg; //Temporary storage
  byte Reg7;  //Si570 register 7
  byte Reg8;  //Si570 register 8
  byte Reg9;  //Si570 register 9
  byte Reg10; //Si570 register 10
  byte Reg11; //Si570 register 11
  byte Reg12; //Si570 register 12
  float RFREQ;  //Si570 RFREQ variable
  int RFREQint; //Integer part of RFREQ
  float RFREQfract; //Fractional part of RFREQ
  float RFREQmod;   //Fractional part of RFREQ multiplied by 2^28
  unsigned long RFREQmodint;  //Integer part of the multiplied fractional part of RFREQ

  HsDiv = 11; //Set HsDiv to highest value

//DividerCalc determines the correct value for Hs_Div
  DividerCalc();
  do  {
    if (Fdco > FdcoUpper || Fdco < FdcoLower)  {
       HsDiv = HsDiv - 1;
        if (HsDiv == 10 || HsDiv == 8)  {
          HsDiv = HsDiv -1;
        }

       DividerCalc();
     }
  } while (Fdco > FdcoUpper || Fdco < FdcoLower);
     switch(HsDiv)  {
    case 4:
      HsDivReg = 0;
      break;
    case 5:
      HsDivReg = 1;
      break;
    case 6:
      HsDivReg = 2;
      break;
    case 7:
```

```
          HsDivReg = 3;
          break;
        case 9:
          HsDivReg = 5;
          break;
        case 11:
          HsDivReg = 7;
          break;
      }

    //  Merge bits 6-2 of NiReg as LSbits with bits 2-0 of HsDivReg as MSbits and place in register 7

      N1Reg = N1 - 1;   //Calculate the register N1 value
      TempReg = HsDivReg << 5;   //Set the register HsDivmod[2:0] value and shift to most significant part of the byte
      Reg7 = TempReg + ((N1Reg >> 2) & 0x1F);    //Concatinate HsDivmod[2:0] and N1mod[2:6} into register 7

    //  Calculate RFREQ and merge bits 37-32 as LSbits with bits 1-0 of NiReg as MSbits and place in register 8

      RFREQ = Fdco / Fxtal;   //Calculate RFREQ
      RFREQint =  RFREQ;    //Get the integer part of RFREQ
      RFREQfract = RFREQ - RFREQint;   //Get the fractional part of RFREQ
      RFREQmod = RFREQfract * 268435456;  //Multiply the fractional part of RFREQ by 2^28
      RFREQmodint =  RFREQmod;  //Integer part of the fractional part of RFREQ*2^28
      String bsRFREQmodint = String(RFREQmodint,BIN); //Convert the multiplied fraction of RFREQ into binary
      bsRFREQmodint = stringPad(28, bsRFREQmodint); //Fill string so it is a 28 bit binar number
      String bsRFREQint = String(RFREQint,BIN); //Convert the integer portion of RFREQ to binary
      bsRFREQint = stringPad(10, bsRFREQint);  //Fill string so that it is a 10 bit binary number
      bsRFREQint += bsRFREQmodint;  //Concatenate the integer and fractional parts
      String bsRFREQ = bsRFREQint; //Save REREQ in binary
      Reg8 = (N1Reg & 3) << 6;    //Place N1mod[1:0] into the most significant part of register 8
      String bsShiftedN1 = String(Reg8,BIN);  //Convert the shifted part of N1 in register 8 to binary
      bsShiftedN1 = stringPad(8, bsShiftedN1); //Fill string so that it is a 8 bit binary number
      String bsUpperRFREQ = bsRFREQ.substring(0,6); //Bits 37-32 of RFREQ
      String bsUbsShiftedN1 = bsShiftedN1.substring(0,2);
      String bsReg8 = bsUbsShiftedN1 += bsUpperRFREQ;
      Reg8 = bindec(bsReg8);
      String bsReg9 = bsRFREQ.substring(6,14);
      Reg9 = bindec(bsReg9);
      String bsReg10 = bsRFREQ.substring(14,22);
      Reg10 = bindec(bsReg10);
      String bsReg11 = bsRFREQ.substring(22,30);
      Reg11 = bindec(bsReg11);
      String bsReg12 = bsRFREQ.substring(30,38);
      Reg12 = bindec(bsReg12);
```

```
  SiRegOut[0] = Reg7;
  SiRegOut[1] = Reg8;
  SiRegOut[2] = Reg9;
  SiRegOut[3] = Reg10;
  SiRegOut[4] = Reg11;
  SiRegOut[5] = Reg12;
}

//Pad the string with zeros from the left
String stringPad(int FillNumber, String stringTemp) {
  String result;
  String stringZero = "0";
  while(stringTemp.length() < FillNumber ) {
    stringZero += stringTemp;
    stringTemp = stringZero;
    stringZero = "0";
   }
    result = stringTemp;
    return result;
 }

//Converts an 8 digit binary number to decimal
int bindec(String parsedString) {
  int result;
  int y;
  int j;
  int i;
  int z = 0;
  String string1;
  for(i = 8; i > 0 ; i--) {
    string1 = parsedString.substring(i-1,i);  //Pick out LS digit of binary number
    if(string1 == "1") {
      j = 8 - i;  //Set the power of 2 if digit not equal to 0
      switch(j) {
        case 0:   // 2^1 = 2
          y = 1;
          break;
        case 1:   // 2^1 = 2
          y = 2;
          break;
        case 2:   // 2^1 = 4
          y = 4;
          break;
        case 3:   // 2^1 = 8
          y = 8;
          break;
        case 4:   // 2^1 = 16
          y = 16;
```

```
        break;
      case 5:   // 2^1 = 32
        y = 32;
        break;
      case 6:   // 2^1 = 64
        y = 64;
        break;
      case 7:   // 2^1 = 128
        y = 128;
        break;
      default:
      break;

    }

    z = z + y;  //Sum the results
   }
  }
 result = z;
 return result;
}

//Converts a 2 digit hexadicmal number to decimal
int hexdec(String parsedString) {
 int result;
 int y;
 String string1;
 string1 = parsedString.substring(1);  //pick up the least significant digit of hex number
 int x = string1.toInt();  //if the digit is a letter set x = 0 else convert x to an integer
  if (string1 == "a") {

    x = 10;
  }
  else if (string1 == "b") {

    x = 11;
  }

  else if (string1 == "c") {

    x = 12;
  }
  else if (string1 == "d")  {

    x = 13;
  }
  else if (string1 == "e") {
```

```
    x = 14;
   }
   else if (string1 == "f") {

    x = 15;
   }
   y = x;
  string1 = parsedString.substring(1,0);  //pick up the most significant digit of the hex number
  x = string1.toInt();  //if the digit is a letter set x = 0 else convert x to an integer
   if (string1 == "a")     {
    x = 10;
   }
   else if (string1 == "b")
   {
    x = 11;
   }

   else if (string1 == "c")
   {
    x = 12;
   }
   else if (string1 == "d")
   {
    x = 13;
   }
   else if (string1 == "e")
   {
    x = 14;
   }
   else if (string1 == "f")
    {
    x = 15;
   }
  x = (16 * x) + y;
   result = x;

  return result;
}

//send a STOP to SI570
void StopTransfer() {
  digitalWrite(SDAT,LOW);
  digitalWrite(SCLK,HIGH);
  digitalWrite(SDAT,HIGH);
}

//pick up acknowledge bit
void AckGet() {
```

```
  byte b;
//  digitalWrite(LED,LOW);
  pinMode(SDAT,INPUT);
  b = digitalRead(SDAT);
  digitalWrite(SCLK,HIGH);
  digitalWrite(SCLK,LOW);
  if(b != 0) {
 //   digitalWrite(LED,HIGH);
NoAck: goto NoAck;
  }
//  digitalWrite(LED,LOW);
  pinMode(SDAT,OUTPUT);
}

void Trigger() {
//  digitalWrite(TrgPls,HIGH);
//  digitalWrite(TrgPls,LOW);
}

//Read registers 7 through 12 of Si570 and prints them out
void ReadSi570() {
 byte b;
 StartTransfer();
 shiftOut(SDAT,SCLK,MSBFIRST,170);
 AckGet();
 shiftOut(SDAT,SCLK,MSBFIRST,7);
 AckGet();
 StartTransfer();
 shiftOut(SDAT,SCLK,MSBFIRST,171);
 AckGet();
 pinMode(SDAT,INPUT);
 for(int i = 0; i < 6; i++)  {
   SiRegIn[i] = shiftIn(SDAT,SCLK,MSBFIRST);
   pinMode(SDAT,OUTPUT);
   digitalWrite(SDAT,LOW);
   digitalWrite(SCLK,HIGH);
   digitalWrite(SCLK,LOW);
   pinMode(SDAT,INPUT);
 }
 pinMode(SDAT,OUTPUT);
 digitalWrite(SDA,HIGH);
 StopTransfer();
 for(int i = 0; i < 6; i++) {
 }
}

//Freezes the DCO
void FreezeDCO() {
```

```
  StartTransfer();  //Start data transfer
  shiftOut(SDAT,SCLK,MSBFIRST,170); //Address Si570 0x55 times 2 for write bit
  AckGet();
  shiftOut(SDAT,SCLK,MSBFIRST,137); //DCO control register address
  AckGet();
  shiftOut(SDAT,SCLK,MSBFIRST,16);  //bit 4 = 1 to freeze DCO
  AckGet();
  StopTransfer();
}

//Unfreezes the DCO and alerts that there is a new frequency outside the limits of +/- 3.5 MHz
void UnFreezeDCO() {
  StartTransfer();  //Start data transfer
  shiftOut(SDAT,SCLK,MSBFIRST,170); //Address Si570 0x55 times 2 for write bit
  AckGet();
  shiftOut(SDAT,SCLK,MSBFIRST,137); //DCO control register address
  AckGet();
  shiftOut(SDAT,SCLK,MSBFIRST,0); //Unfreeze DCO
  AckGet();
  StopTransfer();
  StartTransfer();  //Start data transfer
  shiftOut(SDAT,SCLK,MSBFIRST,170); //Address Si570 0x55 times 2 for write bit
  AckGet();
  shiftOut(SDAT,SCLK,MSBFIRST,135); //Memory control register address
  AckGet();
  shiftOut(SDAT,SCLK,MSBFIRST,64);  //Signal new frequency applied
  AckGet();
  StopTransfer();
}

//Freezes the memory registers
void FreezeM() {
  StartTransfer();  //Start data transfer
  shiftOut(SDAT,SCLK,MSBFIRST,170); //Address Si570 0x55 times 2 for write bit
  AckGet();
  shiftOut(SDAT,SCLK,MSBFIRST,135); //Memory control register address
  AckGet();
  shiftOut(SDAT,SCLK,MSBFIRST,32);  //Prevent frequency glitches while changing RFREQ (freeze M)
  AckGet();
  StopTransfer();
}

//Unfreezes the mememor registers
void UnFreezeM() {
  StartTransfer();  //Start data transfer
  shiftOut(SDAT,SCLK,MSBFIRST,170); //Address Si570 0x55 times 2 for write bit
  AckGet();
  shiftOut(SDAT,SCLK,MSBFIRST,135); //Memory control register address
```

```
  AckGet();
  shiftOut(SDAT,SCLK,MSBFIRST,0); //Unfreeze M
  AckGet();
  StopTransfer();
}

//Set frequency is used when the frequency change is more than +/- 3.5 MHz
void SetFreq(){
  FreezeDCO();
  FreezeM();
  RegCalc();
  StartTransfer();  //Start data transfer
  shiftOut(SDAT,SCLK,MSBFIRST,170); //Address Si570 0x55 times 2 for write bit
  AckGet();
  shiftOut(SDAT,SCLK,MSBFIRST,7); //Address first frequency control register
  AckGet();
  for(int i = 0; i < 6; i++)  {
    shiftOut(SDAT,SCLK,MSBFIRST,SiRegOut[i]);  //Load frequency control registers
    AckGet();
  }
  StopTransfer();
  UnFreezeM();
  UnFreezeDCO();
  delay(20);  //Delay 20 msec for changes to take place
}

// Change frequency is used when the frequency change is within +/- 3.5 MHz
void ChangeFreq() {
  RegCalc();
  StartTransfer();  //Start data transfer
  shiftOut(SDAT,SCLK,MSBFIRST,170); //Address Si570 0x55 times 2 for write bit
  AckGet();
  shiftOut(SDAT,SCLK,MSBFIRST,7); //Address first frequency control register
  AckGet();
  for(int i = 0; i < 6; i++)  {
    shiftOut(SDAT,SCLK,MSBFIRST,SiRegOut[i]);  //Load frequency control registers
    AckGet();
  }
  StopTransfer();
}

void TriState() {
  digitalWrite(SDAT,LOW);
  pinMode(SDAT,INPUT);
}
void TestI2C(){
  Trigger();
  StartTransfer();
```

```
  shiftOut(SDAT,SCLK,MSBFIRST,170);
  AckGet();
  shiftOut(SDAT,SCLK,MSBFIRST,135);
  AckGet();
  shiftOut(SDAT,SCLK,MSBFIRST,0);
  AckGet();
  StopTransfer();
//  ReadSi570();
}

//Initialize LCD
void InitializeLCD()  {
  pinMode(SDAT, OUTPUT); //LCD Data
  pinMode(SCLK, OUTPUT);  //LCD Clock
  pinMode(LCDEnable, OUTPUT); //LCD Enable
  digitalWrite(LCDEnable,HIGH); //Initialize LCD Enable
  digitalWrite(SCLK,HIGH);  //Initialize LCD Clock
  digitalWrite(SDAT,LOW); //Initialize LCD Data
  interrupts();
  delay(100);
  ClearScreen();
  BlinkingCursorOff();
  UnderlineOff();
}

void WriteLCD(int LCDVal)  {
  digitalWrite(LCDEnable,LOW);
  digitalWrite(SCLK, HIGH);
  for(int i = 8; i > 0; i--) {
    int b = bitRead(LCDVal, i-1);
    if(b == 1)  {
      digitalWrite(SDAT,HIGH);
      }
    else if(b == 0)  {
      digitalWrite(SDAT,LOW);
    }
    digitalWrite(SCLK,LOW);
    digitalWrite(SCLK,HIGH);
  }
  digitalWrite(LCDEnable, HIGH);
}

void LCDWriteString(String LCDString)  {
  int x;
  int LCDVal;
  x = LCDString.length();
  for(int i = 0; i < x; i++) {
    LCDVal = LCDString.charAt(i);
```

```
   WriteLCD(LCDVal);
  }
}

//Displays a floating point number with a certain number of decimal places
void WriteFloatLCD(float n, int d) {
  String ThisString = String(n,d);  //Convert to string with d decimal places
  LCDWriteString(ThisString); //Display it
}

//Displays an integer number
void WriteIntLCD(int n) {
  String ThisString = String(n);  //Convert to string
  LCDWriteString(ThisString); //Display it
}

//Diplays an unsigned integer
void WriteUIntLCD(unsigned int n) {
  String ThisString = String(n);  //Convert to string
  LCDWriteString(ThisString); //Display it
}

//Displays an unsigned long
void WriteULongLCD(unsigned long n) {
  String ThisString = String(n);  //Convert to string
  LCDWriteString(ThisString); //Display it
}

//Turns the display on
void DisplayOn()  {
  WriteLCD(254);  //Command prefix
  WriteLCD(65); //Turn diplay on
  delayMicroseconds(100);
}

//Turns the diplay off
void DisplayOff()  {
  WriteLCD(254);  //Command prefix
  WriteLCD(66); //Turn diplay off
  delayMicroseconds(100);
}

//Sets the cursor to the disired position
//Line 1:  0 - 19
//Line 2: 20 - 39
//Line 3: 40 - 59
//Line 4: 60 - 79
void SetCursor(int Pos, int Line) {
```

```
  int CursorPosition;
  WriteLCD(254);  //Command prefix
  WriteLCD(69);
  if(Line == 0) CursorPosition = Pos;
  else if(Line == 1) CursorPosition = Pos + 20;
  else if(Line == 2) CursorPosition = Pos + 40;
  else if(Line == 3) CursorPosition = Pos + 60;
  if((CursorPosition >= 20) && (CursorPosition <= 39)) CursorPosition = CursorPosition + 44;
  else if((CursorPosition >= 40) && (CursorPosition <= 59)) CursorPosition = CursorPosition - 20;
  else if((CursorPosition >= 60) && (CursorPosition <= 79)) CursorPosition = CursorPosition + 24;
  WriteLCD(CursorPosition);
  delayMicroseconds(100);
}


//Sets the cursor to home (location 0)
void CursorHome() {
  WriteLCD(254);  //Command prefix
  WriteLCD(70); //Turn diplay off
  delayMicroseconds(1500);
}

//Underlines the cursor
void UnderlineOn()  {
  WriteLCD(254);  //Command prefix
  WriteLCD(71); //Turn underline on
  delayMicroseconds(1500);
}

void UnderlineOff()  {
  WriteLCD(254);  //Command prefix
  WriteLCD(72); //Turn underline off
  delayMicroseconds(1500);
}

//Move the cursor left one place
void MoveCursorLeft() {
  WriteLCD(254);  //Command prefix
  WriteLCD(73); //Move cursor left
  delayMicroseconds(100);
}

//Move the cursor right one place
void MoveCursorRight() {
  WriteLCD(254);  //Command prefix
  WriteLCD(74); //Move cursor right
  delayMicroseconds(100);
}
```

```
//Turns the blinking character on
void BlinkingCursorOn() {
  WriteLCD(254);  //Command prefix
  WriteLCD(75); //Turn blinking cursor on
  delayMicroseconds(100);
}

//Turns the blinking character off
void BlinkingCursorOff()  {
  WriteLCD(254);  //Command prefix
  WriteLCD(76); //Turn blinking cursor off
  delayMicroseconds(100);
}

//Back space on place
void BackSpace()  {
  WriteLCD(254);  //Command prefix
  WriteLCD(78); //Back space
  delayMicroseconds(100);
}

//Clears the screen
void ClearScreen()  {
  WriteLCD(254);  //Command prefix
  WriteLCD(81); //Clear screen
  delayMicroseconds(2000);
}

//Sets the contrast between 1 and 50
void SetContrast(int c) {
  if((c < 1 ) ||  (c > 50)) c = 50; //Set to maximum if out of bounds
  WriteLCD(254);  //Command prefix
  WriteLCD(82); //Sets the contrast according to value c
  WriteLCD(c);  //Contrast set
  delayMicroseconds(500);
}




//Displays firmware number
void DisplayFirmWare()  {
  WriteLCD(254);  //Command prefix
  WriteLCD(112);  //Display it
}

//Loads the attenuator
```

```
void Attenuator(int AttValu1, int AttValu2)  {
 digitalWrite(AttLatch1, LOW);
 shiftOut(SDAT, SCLK, LSBFIRST, AttValu1);
 digitalWrite(AttLatch1, HIGH);
 digitalWrite(AttLatch1, LOW);

 digitalWrite(AttLatch2, LOW);
 shiftOut(SDAT, SCLK, LSBFIRST, AttValu2);
 digitalWrite(AttLatch2, HIGH);
 digitalWrite(AttLatch2, LOW);
}

void SelectPar()  {
//Blank parameter indicators
  SetCursor(0, 1);
  LCDWriteString(" ");
  SetCursor(0, 2);
  LCDWriteString(" ");
  SetCursor(0, 3);
  LCDWriteString(" ");

//Select paraameter to indicate a change
  switch(ParNum)  {
    case 1:
      ParSelCol = 0;
      ParSelRow = 1;
      SetCursor(ParSelCol ,ParSelRow); //Set cursor to frequency change
      LCDWriteString(">");   //Frequency parameter
    break;
    case 2:
      ParSelCol = 0;
      ParSelRow = 2;
      SetCursor(ParSelCol ,ParSelRow); //Set cursor to RF level
      LCDWriteString(">");  //RF Level parameter
      UnderlineOff();
    break;
    case 3:
      ParSelCol = 0;
      ParSelRow = 3;
      SetCursor(ParSelCol ,ParSelRow); //Set cursor to External Attenuator adjust
      LCDWriteString(">");  //RF PWR adjustment
      UnderlineOff();
    break;
  }

}

//Sets the RF power level in dBM
```

```
void RFPwrSet()  {

}

//*Calculates the attenuation needed get the RF level down
//to 0 dBM including filter and attenuator insertion losses
void RFCorrection() {


//calculate the loss vs frequency here and save as a floating point number, FRFadj
int Delta;  //Fraction of a dB to be added

switch (Band)
{
//Fdisplay:  500000 - 999999
  case 1:
  if (Fdisplay < 880000)
  {
    FRFadj = 6;
  }
  else if ((Fdisplay >= 880000) && (Fdisplay < 980000))
  {
    FRFadj = 5.75;
  }
  else if ((Fdisplay >= 980000) && (Fdisplay < 999999))
  {
    FRFadj = 5.5;
  }
  if (Rdisplay == 10)
  {
//    FRFadj = FRFadj - 0.5;
  }
  break;

//Fdisplay: 1000000 - 1999999
  case 2:
  FRFadj = 6;
  break;

//Fdisplay: 2000000 - 3999999
  case 3: //Fdisplay: 2000000 - 3999999
  if (Fdisplay < 2500000)
  {
    FRFadj = 6;
  }
  else if ((Fdisplay >= 2500000) && (Fdisplay < 2740000))
  {
    FRFadj = 5.75;
```

```
  }
  else if ((Fdisplay >= 2740000) && (Fdisplay < 3400000))
  {
   FRFadj = 6;
  }
  else if ((Fdisplay >= 3400000) && (Fdisplay < 3580000))
  {
   FRFadj = 5.75;
  }
  else if ((Fdisplay >= 3580000) && (Fdisplay < 3800000))
  {
   FRFadj = 5.5;
  }
   else if ((Fdisplay >= 3800000) && (Fdisplay < 3999999))
  {
   FRFadj = 5.75;
  }
  break;

//Fdisplay: 4000000 - 7999999
  case 4: //Fdisplay: 4000000 - 7999999
  if ((Fdisplay >= 4000000) &&(Fdisplay < 5300000))
  {
   FRFadj = 6;
  }
  else if ((Fdisplay >= 5300000) && (Fdisplay < 6840000))
  {
   FRFadj = 6.5;
  }
  else if ((Fdisplay >= 6840000) && (Fdisplay < 7900000))
  {
   FRFadj = 6;
  }
  break;

//Fdisplay: 8000000 - 15999999
  case 5: //Fdisplay: 8000000 - 15999999
  if ((Fdisplay >= 8000000) &&(Fdisplay < 12800000))
  {
  FRFadj = 5.5;
  }
  break;
  case 6: //Fdisplay: 12800000 - 31999999
  if (Fdisplay < 26500000)
  {
   FRFadj = 5.25;
  }
  else if ((Fdisplay >= 26500000) && (Fdisplay < 30200000))
```

```
 {
  FRFadj = 5.25;
 }
 else if ((Fdisplay >= 30200000) && (Fdisplay < 31999999))
 {
  FRFadj = 5;
 }
 break;
 case 7: //Fdisplay: 32000000 - 63999999
 FRFadj = 5.25;
 if ((Fdisplay >= 53000000) && (Fdisplay < 56000000))
 {
  FRFadj = 4.75;
 }
 else if ((Fdisplay >= 56000000) && (Fdisplay < 60000000))
 {
  FRFadj = 4.25;
 }
 else if ((Fdisplay >= 60000000) && (Fdisplay < 63000000))
 {
  FRFadj = 3.75;
 }
 else if ((Fdisplay >= 63000000) && (Fdisplay < 63999999))
 {
  FRFadj = 3.25;
 }

 break;
 case 8: //Fdisplay: 64000000 - 128000000
  if ((Fdisplay >= 64000000) && (Fdisplay < 104000000))
  {
   FRFadj = 5;
  }
 else if ((Fdisplay >= 104000000) && (Fdisplay < 127999999))
 {
/*From 104 t0 128 MHz, the 0-dBm amplitude varies parabolically, so a nonlinear least squares
  was used to calculate the polynomial coefficients*/
  int fd = Fdisplay / 1000000;
  float delta;
  float a = -.00935;
  float b = 2.171;
  float c = -124.2;
  delta = a*fd*fd + b*fd + c;
  FRFadj = 5 - delta;
 }

 break;
```

```
}
//  FRFadj = 6;
FRFadj = FRFadj - PAdisplay;

//dB attenuation to get 0 dBm after filter and attenuator losses
float FineAdjust = FRFadj - int (FRFadj); //Get the fractional part of the attenuation value

if ((FineAdjust >= 0) && (FineAdjust < 0.125))
{
  Delta = 0;
}
else if ((FineAdjust >= 0.125) && (FineAdjust < 0.375))
{
  Delta  = 1; //Adjust the attenuation by 0.25 db
}
else if ((FineAdjust >= 0.375) && (FineAdjust < 0.625))
{
   Delta = 2; //Adjust the attenuation by 0.5 db
}
else if ((FineAdjust >= 0.625) && (FineAdjust < 0.875))
{
  Delta = 3;  //Adjust the attenuation by 0.75 db
}
else if ((FineAdjust >= 0.875) && (FineAdjust < 1))
{
  Delta = 0;  //Adjust the attenuation by 1.0 db
  FRFadj = FRFadj + 1;
}
IRdisplay = int(Rdisplay);  //Convert to integer
IRdisplay = IRdisplay + int (FRFadj);  //Compute the integer attenuator value


ATTvalu1 = IRdisplay + Delta;  //Compute the final attenuation value for attenuator #1 and convert
to integer
ATTvalu2 = IRdisplay;  //Compute the final attenuation value for attenuator #2, #3, and #4 and
convert to integer

//Adjust for overflow in attenuator #1
if (ATTvalu1 > 127)
{
  ATTvalu1 = ATTvalu1 - 3;
  ATTvalu2 = ATTvalu2 + 1;
}

//Serial.print(FRFadj);
//Serial.print("  ");
//Serial.print(Delta);
//Serial.print("  ");
```

```
//Serial.print(ATTvalu1);
//Serial.print("  ");
//Serial.println(ATTvalu2);

Attenuator(ATTvalu1, ATTvalu2); //Set the attenuator value

 }
```