```
#define TOGGLE(port,bit) (port ^= (1<<bit))
#define StateA 11 //Assign optoencoder state A to pin 11
#define StateB 12 //Assign optoencoder state B to pin 12
#define Run 13  //Assign Run to pin 13
#define SDAT A5 //Assign A5 to serial data
#define SCLK A4 //Assign A4 to serial clock
#define LCDEnable A2  //Assign A2 to LCD enable
#define Trigger A3  //Assign A3 to HP8568B sweep trigger
#define AnalogPin A0  //Assign A0 to the switch reader
const double DDSFactor = 4294967296.0;  //AD9851 constant
const double ClockFrequency = 180000000.0;  //AD9851 clock frequency
double FrequencyFactor; //factor times frequency to get AD9851 tuning word
unsigned long Frequency; //AD9851 desired frequency
unsigned long StartFrequency = 100000;  //Sweep start frequency
unsigned long StartFrequencyW;  //save wide bandwidth settings
String StartFrequencyString;
String StartFrequencyDisplay;
unsigned long StopFrequency = 42000000; //Sweep stop frequency
unsigned long StopFrequencyW; //save wide bandwidth settings
String StopFrequencyDisplay;
String StopFrequencyString;
unsigned long FrequencySpan; //Frequency span in Hz
String FrequencySpanString;
String FrequencySpanDisplay;
String FrequencyDisplay;
unsigned long  CenterFrequency = 10000000;
String CenterFrequencyString;
String CenterFrequencyDisplay;
unsigned long  Span = 28000;
String SpanString;
String SpanDisplay;
unsigned long SweepTime = 700;  //HP8568B sweep time in milliseconds
String SweepTimeString;
String SweepTimeDisplay;
unsigned long FrequencySteps; //Number of frequency steps in a sweep
unsigned long DDSWord;  //AD9851 tuning word
unsigned long DeltaFrequency;  //Frequency offset for each tuning step
unsigned long FreqPerMicro;
unsigned long FrequencyPer30;
unsigned long FrequencyOffset;
unsigned long i;  //Loop counter
unsigned long InterruptCount = 0; //Interrupt counts for a whole sweep
byte Flag = LOW;  //Frequencies ar incremented when this flag is high for a sweep period
byte LoadRun = LOW; //Load state is HIGH.  Run state is LOW
byte CW = HIGH;  //Rotation flag
byte ChangeParameter = HIGH;   //Indicates a parameter change
byte Astate;  //Opto state A
byte AlastState;  //Opto last state A
byte Bstate;  //Opto state B
byte BlastState;  //Opto last state B
byte ParameterSelect = 0;
byte MSDigit = LOW;
byte LSDigit = LOW;
```

```
byte BandWidth = HIGH;
byte MSBPulse = LOW;
byte LSBPulse = LOW;
int counter = 0;  //Opto counter
int AnalogVal = 0;
unsigned int StrtFreqDig = 12;
unsigned int StopFreqDig = 12;
unsigned int CentFreqDig = 12;
unsigned int SpanFreqDig = 13;
unsigned int SweepTimeDig = 14;




void setup()
{
  pinMode(Trigger, OUTPUT); //HP8658B Sweep Trigger output
  pinMode(Run, INPUT); //Run/Load switch (LoadRun == LOW => RUN)
  pinMode(SDAT, OUTPUT);  //LCD serial data
  pinMode(SCLK, OUTPUT);  //LCD serial clock
  pinMode(LCDEnable, OUTPUT); //LCD enable


FrequencyFactor = DDSFactor / ClockFrequency;

//Pin 8 is AD9851 W_CLK
//Pin 9 is AD9851 FQ_UD
//Pin 10 is AD9851 RESET
//B00000001 => that Pin 0 of port D is high
//B00000010 => that Pin 1 of port D is high
//Therefore Pin 0 is LSB and pin 7 is MSB
//On PORTB LSB is Pin 8
  DDRD = 0xFF;  //PORT D for output
  DDRB = B00000111; //Set up PORT B, pin 8, pin 9, and pin 10 for output
  PORTB = B00000000;
  PORTB = B00000100;  // Pin 10 pulsed (Reset DDS)
  PORTB = B00000000;
  delay(500); //Delay for LCD
  InitializeLCD();  //Initialize the LCD
  SetContrast(50);  //Set the LCD contrast to maximum
  SetCursor(0, 0);  //Set cursor
  LCDWriteString(">");  //Place indicator in first position initially
  StartFrequencyW = StartFrequency; //Remember wide bandwidth settings
  StartFrequencyString = String(StartFrequency);
  StopFrequencyW = StopFrequency; //Remember wide bandwidth settings
  StopFrequencyString = String(StopFrequency);
  FrequencySpanString = String(FrequencySpan);
  CenterFrequencyString = String(CenterFrequency);
  SpanString = String(Span);
  Astate = digitalRead(StateA);
  AlastState = Astate;
  Bstate = digitalRead(StateB);
```

```
  BlastState = Bstate;
//  Serial.begin(9600);









}

void loop()
{
  LoadRun = digitalRead(Run);
  if(LoadRun == HIGH)
  {
    ResetTimers();  //Reset the timers so the delay routine works
    OptoEncoder();  //Counter increases or decreases when the optoencoder is rotated
    Switches(); //Reads the switches
    PrepareDisplay();
    ChangeLCD(BandWidth);  //Cahnges the LCD values
    UnderLine();
    FrequencyParameters();  //Calculate the frequency parameters
    SweepTimeString = String(SweepTime);
  }
  else
  {
    SetUpTimers();  //Set up TIMER1 for 100 usecs and disable TIMER0 and TIMER2
    TrackingGenerator();
  }
}

//SetUpTimers Sets up TIMER1 for 100 usecs and disable TIMER0 and TIMER2
void SetUpTimers()
{
  noInterrupts(); //Turn off the interrupts
  TIMSK2 = 0; //Turn off Timer 2
  TIMSK0 = 0; //Turn off Timer 0
  TCCR1A = 0; //Reset Timer 1 control register
  TCCR1B = 1; //Set for no prescaler
  OCR1A = 1550; //Set compare register for 100 usec
  TCNT1 = 0;  //Reset Timer 1
  TIMSK1 = 2; //Enable Timer1 compare interrupt
}


//Re enable interrupts for Serial Printing and Delay
void ResetTimers()
{
  noInterrupts(); //Re enable interrupts for Serial Printing and Delay
  TIMSK0 =1;
  TIMSK1 = 0;
```

```
 interrupts();
}

//Delay without timer interrupts
void Delay(unsigned long DelayConstant)
{
 for(i = 0; i < DelayConstant; i++)
 {
 delayMicroseconds(1000);
 }
}

ISR(TIMER1_COMPA_vect)  //Timer interrupt routine
{
  TCNT1 = 0;  //Reset interrupt
  if(Flag == HIGH) //Load frequencies while sweep is in progress
  {
    DDSLoad();  //Load the new frequency information to the AD9851
    InterruptCount++; //Increment the interrupt count
}
}

void TrackingGenerator()
{
 LoadRun = digitalRead(Run);
 if(LoadRun == LOW)
 {
  do
  {
    digitalWrite(Trigger, HIGH);  //Start a sweep on the HP8568B
    Flag = HIGH;  //Flag the interrupt routine that a sweep has started
    interrupts();   //Turn on the interrupts
  }while (InterruptCount <= FrequencySteps);  //Keep sweep going until all frequencies have been swept
    Flag = LOW; //Flag the interrupt routine that the sweep is complete
    noInterrupts();
    InterruptCount = 0; //Reset the interrupt count
    digitalWrite(Trigger, LOW); //Stop the HP8568B sweep
    Delay(300);
 }
}


//Calculate the AD9851 frequency paramaters
void FrequencyParameters()
{
 if(BandWidth == LOW)
 {
  StartFrequency = CenterFrequency - (Span / 2);
  StopFrequency = CenterFrequency + (Span / 2);
 }
 else if (BandWidth == HIGH)
 {
  StartFrequency = StartFrequencyW; //Restore wide bandwidth settings
```

```
      StopFrequency = StopFrequencyW;  //Restore wide bandwidth settings
     }
     FrequencySpan = StopFrequency - StartFrequency; //Compute the frequency span
     FrequencySteps = (SweepTime * 10);  //Compute the number of frequency steps in a sweep


     if(DeltaFrequency <= 0) { DeltaFrequency = 1; }
    }

    //Load the new frequency into the AD9851
    void DDSLoad()
    {
    //    TOGGLE(PORTC, 3);
        Frequency = StartFrequency + ((FrequencySpan / FrequencySteps) * InterruptCount);
        DDSWord = Frequency * FrequencyFactor;  //Calculate DDS word
        PORTD = B00000001;  //DDS control register
        PORTB = B00000001; //Pin 8 high (AD9851 W_CLK)
        PORTB = B00000000; //Pin 8 low
        PORTD = DDSWord >> 24;  //MSByte of DDS Frequeny Word
        PORTB = B00000001; //Pin 8 high
        PORTB = B00000000; //Pin 8 low
        PORTD = lowByte(DDSWord >> 16);
        PORTB = B00000001; //Pin 8 high
        PORTB = B00000000; //Pin 8 low
        PORTD = lowByte(DDSWord >> 8);
        PORTB = B00000001; //Pin 8 high
        PORTB = B00000000; //Pin 8 low
        PORTD = lowByte(DDSWord); //LSByte of DDS Frequency Word
        PORTB = B00000001; //Pin 8 high
        PORTB = B00000000; //Pin 8 low
        PORTB = B00000010;  //Pin 9 pulsed ((AD9851 FQ_UD)
        PORTB = B00000000;

    }


    //StartFrequencyCorrection does 20 usecs worth of frequency correction assuming a 30 usec delay from
    //sweep start and a 50 usec interrupt period.
    void StartFrequencyCorrection()
    {
     FreqPerMicro = DeltaFrequency / 50; //Frequency change per step divided by step time in microseconds =>
frequency change per microsecond
     FrequencyPer30 = FreqPerMicro * 30;  //Frequency offset for the 30 microsecond delay
     FrequencyOffset = DeltaFrequency - FrequencyPer30;  //Frequency offset for 30 microsecond delay
     Frequency = StartFrequency - FrequencyOffset; //Compensated start frequency
    }

    //********** LCD Routines
**********************************************************************************************
**

    //Initialize LCD initializes the 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();
 }

 //ChangeLCD changes the LCD readings based on bandwidth. Wide bandwidth is HIGH.  Narrow bandwidth is
LOW
 void ChangeLCD(byte BandWidth)
 {
  if(ChangeParameter == HIGH)
  {
   if(BandWidth == HIGH)
   {
      SetCursor(1, 0);
      LCDWriteString("  Wide Bandwidth "); //Diplay bandwidth mode fro wide sweep
      SetCursor(1, 1);
      LCDWriteString("Strt Freq:        ");  //Diplay start frequency label
      SetCursor(11, 1);
      LCDWriteString(StartFrequencyDisplay); //Display start frequency
      SetCursor(1, 2);
      LCDWriteString("Stop Freq:        ");  //Display stop frequency label
      SetCursor(11, 2);
      LCDWriteString(StopFrequencyDisplay);  //Display stop frequency
   }
   else if(BandWidth == LOW)
   {
      SetCursor(1, 0);
      LCDWriteString(" Narrow Bandwidth");  //Diplay bandwidth mode for narrow sweep
      SetCursor(1, 1);
      LCDWriteString("Cent Freq:        ");  //Diplay narrow sweep center frequency label
      SetCursor(11, 1);
      LCDWriteString(CenterFrequencyDisplay);  //Diplay narrow sweep center frequency
      SetCursor(1, 2);
      LCDWriteString("Span (Hz):        ");  //Display narrow sweep span
      SetCursor(11, 2);
      LCDWriteString(SpanDisplay); //Display narrow sweep
   }
  }
  SetCursor(1, 3);  //Set the cursor position
  LCDWriteString("Sweep (ms):       "); //Display sweep time label
  SetCursor(12,3);
  LCDWriteString(SweepTimeDisplay); //Display sweep time
  ChangeParameter = LOW;
 }
```

```
//Do underline
void UnderLine()
{
 if((ParameterSelect == 1) & (BandWidth == HIGH))
 {
  if(LSBPulse == HIGH)
  {
   StrtFreqDig++;  //Increment digit place
   if(StrtFreqDig == 13) { StrtFreqDig = 14; } //Skip over the decimal point
   if(StrtFreqDig > 19) { StrtFreqDig = 19; }  //Stop at the end
   LSBPulse = LOW; //Reset flag
  }
  else if (MSBPulse == HIGH)
  {
   StrtFreqDig--;
   if(StrtFreqDig == 13){ StrtFreqDig = 12; }
   if(StrtFreqDig < 11) { StrtFreqDig = 11; }
   MSBPulse = LOW;
  }
  SetCursor(StrtFreqDig, 1);
  UnderlineOn();
 }
 else if((ParameterSelect == 2) & (BandWidth == HIGH))
 {
  if(LSBPulse == HIGH)
  {
   StopFreqDig++;
   if(StopFreqDig == 13) { StopFreqDig = 14; }
   if(StopFreqDig > 19) { StopFreqDig = 19; }
   LSBPulse = LOW;
  }
  else if (MSBPulse == HIGH)
  {
   StopFreqDig--;
   if(StopFreqDig == 13){ StopFreqDig = 12; }
   if(StopFreqDig < 11) { StopFreqDig = 11; }
   MSBPulse = LOW;
  }
  SetCursor(StopFreqDig, 2);
  UnderlineOn();
 }
 else if((ParameterSelect == 1) & (BandWidth == LOW))
 {
  if(LSBPulse == HIGH)
  {
   CentFreqDig++;
   if(CentFreqDig == 13) { CentFreqDig = 14; }
   if(CentFreqDig > 19) { CentFreqDig = 19; }
   LSBPulse = LOW;
  }
  else if(MSBPulse == HIGH)
  {
   CentFreqDig--;
```

```
    if(CentFreqDig == 13) { CentFreqDig = 12; }
    if(CentFreqDig < 11) { CentFreqDig = 11; }
    MSBPulse = LOW;
   }
   SetCursor(CentFreqDig, 1);
   UnderlineOn();
  }
  else if((ParameterSelect == 2) & (BandWidth == LOW))
  {
   if(LSBPulse == HIGH)
   {
    SpanFreqDig++;
    if(SpanFreqDig > 17) { SpanFreqDig = 17; }
    LSBPulse = LOW;
   }
   else if(MSBPulse == HIGH)
   {
    SpanFreqDig--;
    if(SpanFreqDig < 11) { SpanFreqDig = 11; }
    MSBPulse = LOW;
   }
   SetCursor(SpanFreqDig, 2);
   UnderlineOn();
  }

  else if(ParameterSelect == 3)
  {
   if(LSBPulse == HIGH)
   {
    SweepTimeDig++;
    if(SweepTimeDig > 15) { SweepTimeDig = 15; }
    LSBPulse = LOW;
   }
   else if(MSBPulse == HIGH)
   {
    SweepTimeDig--;
    if(SweepTimeDig < 12) { SweepTimeDig = 12; }
    MSBPulse = LOW;
   }
   SetCursor(SweepTimeDig, 3);
   UnderlineOn();
  }
  else
  {
   UnderlineOff();
  }
}

//PrepareDisplay takes the string value of a parameter and modifies it for display on the LCD
void PrepareDisplay()
{
 String StartString;  //Left part of the string
 String EndString;  //Right part of the string
```

```
int x;
 x = StartFrequencyString.length();  //Get the length of the start frequency string
 if(x == 8)  //This for tens of MHz
 {
   StartString = StartFrequencyString.substring(0, 2); //Get left part of string
   EndString = StartFrequencyString.substring(2, x); //Get right part of string
   StartFrequencyDisplay = StartString + "." + EndString;  //Make composite string
 }
 else if(x == 7) //This for ones of MHz
 {
   StartString = StartFrequencyString.substring(0, 1); //Get left part of string
   EndString = StartFrequencyString.substring(1, x); //Get right part of string
   StartFrequencyDisplay = "0" + StartString + "." + EndString;
 }
 else if(x == 6) //This for tenths of MHz
 {
   StartFrequencyDisplay = "00." + StartFrequencyString;  //Make composite string
 }
//*********
x = StopFrequencyString.length();
if(x == 8)  //This for tens of MHz
{
   StartString = StopFrequencyString.substring(0, 2); //Get left part of string
   EndString = StopFrequencyString.substring(2, x); //Get right part of string
   StopFrequencyDisplay = StartString + "." + EndString;  //Make composite string
}
else if(x == 7) //This for ones of MHz
{
   StartString = StopFrequencyString.substring(0, 1); //Get left part of string
   EndString = StopFrequencyString.substring(1, x); //Get right part of string
   StopFrequencyDisplay = "0" + StartString + "." + EndString;  //Make composite string
}
else if(x == 6) //This for tenths of MHz
{
   StopFrequencyDisplay = "00." + StopFrequencyString;  //Make composite string
}
//*********
x = CenterFrequencyString.length();

if(x == 8)  //This for tens of MHz
{
   StartString = CenterFrequencyString.substring(0, 2); //Get left part of string
   EndString = CenterFrequencyString.substring(2, x); //Get right part of string
   CenterFrequencyDisplay = StartString + "." + EndString;  //Make composite string
}
else if(x == 7) //This for ones of MHz
{
   StartString = CenterFrequencyString.substring(0, 1); //Get left part of string
   EndString = CenterFrequencyString.substring(1, x); //Get right part of string
   CenterFrequencyDisplay = "0" + StartString + "." + EndString;  //Make composite string
}
else if(x == 6) //This for tenths of MHz
{
```

```
   CenterFrequencyDisplay = "00." + CenterFrequencyString;  //Make composite string
 }

//**********
x = SpanString.length();
if(x == 7) { SpanDisplay = SpanString; }
if(x == 6) { SpanDisplay = "0" + SpanString; }
if(x == 5) { SpanDisplay = "00" + SpanString; }
if(x == 4) { SpanDisplay = "000" + SpanString; }
if(x == 3) { SpanDisplay = "0000" + SpanString; }
if(x == 2) { SpanDisplay = "00000" + SpanString; }
if(x == 1) { SpanDisplay = "000000" + SpanString; }
//**********
x = SweepTimeString.length();
if(x == 4) { SweepTimeDisplay = SweepTimeString; }
if(x == 3) { SweepTimeDisplay = "0" + SweepTimeString; }
if(x == 2) { SweepTimeDisplay = "00" + SweepTimeString; }
if(x == 1) { SweepTimeDisplay = "000" + SweepTimeString; }
}

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
}

//************************************** Optoencoder and switch routines
*********************************************


//OptoEncoder reads the optoencoder and changes the count
void OptoEncoder()
{
Astate = digitalRead(StateA); //Read state A
  if(Astate != AlastState)  //Check if state A has changed
  {
    ChangeParameter = HIGH;
    if(Astate == HIGH)  //Do if state A has changed from LOW to HIGH
    {
      if(Bstate == HIGH)  //Flag CW rotation
      {
        CW = HIGH;  //Flag a CW rotation
        ParameterAlter(); //Change the parameter
      }
      else  //Otherwise if state B is LOW, flag a CCW rotation
      {
        CW = LOW;  //Flag a CCW rotation
        ParameterAlter(); //Change the parameter
      }
    }
    if(Astate == LOW)  //Do if state A has changed from HIGH to LOW
    {
      if(Bstate == LOW)  //Flag CW rotation
      {
```

```
      CW = HIGH;  //Flag a clockwise rotation
      ParameterAlter(); //Change the parameter
    }
    else  //Otherwise if state B is HIGH, flag a CCW rotation
    {
      CW = LOW;  //Flag a clockwise rotation
      ParameterAlter(); //Change the parameter
    }
   }
   AlastState = Astate;  //Update the last state of A with the current state of A
  }


  Bstate = digitalRead(StateB); //Read state B
  if(Bstate != BlastState)  //Check if state B has changed
  {
   ChangeParameter = HIGH;
   if(Bstate == HIGH)  //Do if state B has changed from LOW to HIGH
   {
    if(Astate == HIGH)  //Flag CCW rotation
    {
      CW = LOW;  //Flag a CCW rotation
      ParameterAlter(); //Change the parameter
    }
    else  //Otherwise if state A is LOW, flag CW rotation
    {
      CW = HIGH;  //Flag a clockwise rotation
      ParameterAlter(); //Change the parameter
    }
   }

   if(Bstate == LOW)  //Do if state B has changed from HIGH to LOW
   {
    if(Astate == LOW)  //Flag a CCW rotation
    {
      CW = LOW;  //Flag a CCW rotation
      ParameterAlter(); //Change the parameter
    }
    else  //Otherwise if state A is HIGH, decrement the counter
    {
      CW = HIGH;  //Flag a clockwise rotation
      ParameterAlter(); //Change the parameter
    }
   }
   BlastState = Bstate;  //Update the last state of B with the current state of B
  }
}



//ParameterAlter changes the particular parameter
void ParameterAlter()
{
```

```
if(ParameterSelect == 0)  //Switch between wide and narrow bandwidth modes
{
  if(BandWidth == HIGH)
  {
    BandWidth = LOW;
  }
  else
  {
    BandWidth = HIGH;
  }

}
if(ParameterSelect == 1)  //Alter the start  frequency on both wide and narrow bandwidths
{
  if((BandWidth == HIGH) & (CW == HIGH))
  {
    StartFrequency = StartFrequencyW; //Restore wide bandwidth settings
    if(StrtFreqDig == 19) { StartFrequency = StartFrequency + 1; }
    if(StrtFreqDig == 18) { StartFrequency = StartFrequency + 10; }
    if(StrtFreqDig == 17) { StartFrequency = StartFrequency + 100; }
    if(StrtFreqDig == 16) { StartFrequency = StartFrequency + 1000; }
    if(StrtFreqDig == 15) { StartFrequency = StartFrequency + 10000; }
    if(StrtFreqDig == 14) { StartFrequency = StartFrequency + 100000; }
    if(StrtFreqDig == 12) { StartFrequency = StartFrequency + 1000000; }
    if(StrtFreqDig == 11) { StartFrequency = StartFrequency + 10000000; }
    if(StartFrequency > 50000000) { StartFrequency = 50000000; }
    StartFrequencyString = String(StartFrequency);
    StartFrequencyW = StartFrequency; //Save wide bandwidth settings
  }
  else if((BandWidth == HIGH) & (CW == LOW))
  {
    StartFrequency = StartFrequencyW; //Restore wide bandwidth settings
    if(StrtFreqDig == 19) { StartFrequency = StartFrequency - 1; }
    if(StrtFreqDig == 18) { StartFrequency = StartFrequency - 10; }
    if(StrtFreqDig == 17) { StartFrequency = StartFrequency - 100; }
    if(StrtFreqDig == 16) { StartFrequency = StartFrequency - 1000; }
    if(StrtFreqDig == 15) { StartFrequency = StartFrequency - 10000; }
    if(StrtFreqDig == 14) { StartFrequency = StartFrequency - 100000; }
    if(StrtFreqDig == 12) { StartFrequency = StartFrequency - 1000000; }
    if(StrtFreqDig == 11) { StartFrequency = StartFrequency - 10000000; }
    if(StartFrequency <= 100000) { StartFrequency = 100000; }
    StartFrequencyString = String(StartFrequency);
    StartFrequencyW = StartFrequency;
  }
  else if((BandWidth == LOW) & (CW == HIGH))
  {
    if(CentFreqDig == 19) { CenterFrequency = CenterFrequency + 1; }
    if(CentFreqDig == 18) { CenterFrequency = CenterFrequency + 10; }
    if(CentFreqDig == 17) { CenterFrequency = CenterFrequency + 100; }
    if(CentFreqDig == 16) { CenterFrequency = CenterFrequency + 1000; }
    if(CentFreqDig == 15) { CenterFrequency = CenterFrequency + 10000; }
    if(CentFreqDig == 14) { CenterFrequency = CenterFrequency + 100000; }
    if(CentFreqDig == 12) { CenterFrequency = CenterFrequency + 1000000; }
```

```
   if(CentFreqDig == 11) { CenterFrequency = CenterFrequency + 10000000; }
   if(CenterFrequency > 50000000) { CenterFrequency = 50000000; }
   CenterFrequencyString = String(CenterFrequency);
  }
  else if((BandWidth == LOW) & (CW == LOW))
  {
   if(CentFreqDig == 19) { CenterFrequency = CenterFrequency - 1; }
   if(CentFreqDig == 18) { CenterFrequency = CenterFrequency - 10; }
   if(CentFreqDig == 17) { CenterFrequency = CenterFrequency - 100; }
   if(CentFreqDig == 16) { CenterFrequency = CenterFrequency - 1000; }
   if(CentFreqDig == 15) { CenterFrequency = CenterFrequency - 10000; }
   if(CentFreqDig == 14) { CenterFrequency = CenterFrequency - 100000; }
   if(CentFreqDig == 12) { CenterFrequency = CenterFrequency - 1000000; }
   if(CentFreqDig == 11) { CenterFrequency = CenterFrequency - 10000000; }
   if(CenterFrequency <= 100000) { CenterFrequency = 100000; }
   CenterFrequencyString = String(CenterFrequency);

  }

}
if(ParameterSelect == 2)  //Alter the stop frequency on the wide bandwidth
{
  if((BandWidth == HIGH) & (CW == HIGH))
  {
   StopFrequency = StopFrequencyW; //Restore wide bandwidth settings
   if(StopFreqDig == 19) { StopFrequency = StopFrequency + 1; }
   if(StopFreqDig == 18) { StopFrequency = StopFrequency + 10; }
   if(StopFreqDig == 17) { StopFrequency = StopFrequency + 100; }
   if(StopFreqDig == 16) { StopFrequency = StopFrequency + 1000; }
   if(StopFreqDig == 15) { StopFrequency = StopFrequency + 10000; }
   if(StopFreqDig == 14) { StopFrequency = StopFrequency + 100000; }
   if(StopFreqDig == 12) { StopFrequency = StopFrequency + 1000000; }
   if(StopFreqDig == 11) { StopFrequency = StopFrequency + 10000000; }
   if(StopFrequency > 50000000) { StopFrequency = 50000000; }
   StopFrequencyString = String(StopFrequency);
   StopFrequencyW = StopFrequency; //Save wide bandwidth settings
  }
  else if((BandWidth == HIGH) & (CW == LOW))
  {
   StopFrequency = StopFrequencyW; //Restore wide bandwidth settings
   if(StopFreqDig == 19) { StopFrequency = StopFrequency - 1; }
   if(StopFreqDig == 18) { StopFrequency = StopFrequency - 10; }
   if(StopFreqDig == 17) { StopFrequency = StopFrequency - 100; }
   if(StopFreqDig == 16) { StopFrequency = StopFrequency - 1000; }
   if(StopFreqDig == 15) { StopFrequency = StopFrequency - 10000; }
   if(StopFreqDig == 14) { StopFrequency = StopFrequency - 100000; }
   if(StopFreqDig == 12) { StopFrequency = StopFrequency - 1000000; }
   if(StopFreqDig == 11) { StopFrequency = StopFrequency - 10000000; }
   if(StopFrequency <= 100000) { StopFrequency = 100000; }
   StopFrequencyString = String(StopFrequency);
   StopFrequencyW = StopFrequency; //Save wide bandwidth settings
  }
```

```
  else if((BandWidth == LOW) & (CW == HIGH)) //Alter the span on the Narrow bandwidth
 {
  if(SpanFreqDig == 17) { Span = Span + 1; }
  if(SpanFreqDig == 16) { Span = Span + 10; }
  if(SpanFreqDig == 15) { Span = Span + 100; }
  if(SpanFreqDig == 14) { Span = Span + 1000; }
  if(SpanFreqDig == 13) { Span = Span + 10000; }
  if(SpanFreqDig == 12) { Span = Span + 100000; }
  if(SpanFreqDig == 11) { Span = Span + 1000000; }
  if(Span > 1000000) { Span = 1000000; }
  SpanString = String(Span);
 }
 else if((BandWidth == LOW) & (CW == LOW))
 {
  if(SpanFreqDig == 17) { Span = Span - 1; }
  if(SpanFreqDig == 16) { Span = Span - 10; }
  if(SpanFreqDig == 15) { Span = Span - 100; }
  if(SpanFreqDig == 14) { Span = Span - 1000; }
  if(SpanFreqDig == 13) { Span = Span - 10000; }
  if(SpanFreqDig == 12) { Span = Span - 100000; }
  if(SpanFreqDig == 11) { Span = Span - 1000000; }
  if(Span <= 20) { Span = 20; }
  SpanString = String(Span);
 }
}
if(ParameterSelect == 3)
 if(CW == HIGH)
 {
  {
   if(SweepTimeDig == 15) { SweepTime = SweepTime + 1; }
   if(SweepTimeDig == 14) { SweepTime = SweepTime + 10; }
   if(SweepTimeDig == 13) { SweepTime = SweepTime + 100; }
   if(SweepTimeDig == 12) { SweepTime = SweepTime + 1000; }
   if(SweepTime > 9000) {SweepTime = 9000; }
   SweepTimeString = String(SweepTime);
  }
 }
 else if(CW == LOW)
 {
   if(SweepTimeDig == 15) { SweepTime = SweepTime - 1; }
   if(SweepTimeDig == 14) { SweepTime = SweepTime - 10; }
   if(SweepTimeDig == 13) { SweepTime = SweepTime - 100; }
   if(SweepTimeDig == 12) { SweepTime = SweepTime - 1000; }
   if(SweepTime < 1) { SweepTime = 1; }
   SweepTimeString = String(SweepTime);
 }
}



//Switches reads the switches and sets the appropriate flags
void Switches()
{
```

```
 AnalogVal = analogRead(AnalogPin);
 delay(30);
 if((AnalogVal < 400) | (AnalogVal > 420))
 {
  ChangeParameter = HIGH;
  {
   if((AnalogVal > 231) & (AnalogVal < 281)) //Select parameter to manipulate
   {
    ParameterSelect++;  //Advance to next parameter
    if(ParameterSelect > 3) //If last parameter, go back to beginning
    {
     ParameterSelect = 0;  //Back to beginning
     SetCursor(0, 3);  //Select indicator position
     LCDWriteString("");  //Erase indicator
    }
   SetCursor(0, ParameterSelect - 1); //Select previous parameter position
   LCDWriteString(" "); //Erase indicator
   SetCursor(0, ParameterSelect); //Select current parameter position
   LCDWriteString(">"); //Place indicator
   SwitchDelay();
   }
   else if((AnalogVal > 461) & (AnalogVal < 563)) //Move underline to the right for selected parameter
   {
    MSBPulse = HIGH;
    SwitchDelay();
   }
   else if((AnalogVal > 614) & (AnalogVal < 750)) //Move underline to the right for selected parameter
   {
    LSBPulse = HIGH;
    SwitchDelay();
   }
  }
 }
}

void SwitchDelay()
{
 do
 {
  AnalogVal = analogRead(AnalogPin);
 }while((AnalogVal < 400) | (AnalogVal > 420));
}
```