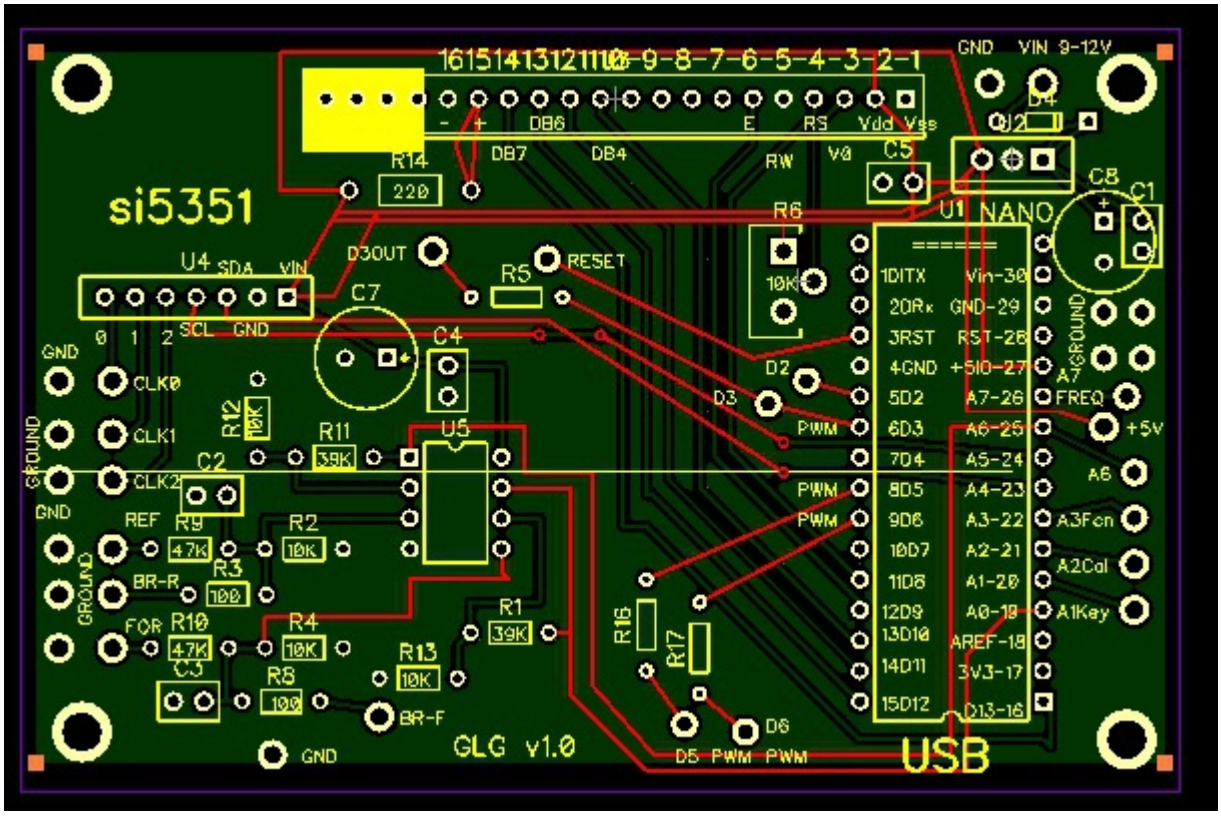
# KX4Z Homebrew Direct Digital Synthesis
# VFO / Display / SWR Measurement Input

Gordon Gibby KX4Z
April 2018




INTRODUCTION

The groundbreaking bitx-series of extraordinarily inexpensive low-power HF single sideband and CW transceivers have made impressive usage of the Si5351 frequency generator chip, and the Arduino microprocessor for frequency, mode, scan, receiver incremental tuning (RIT), integrated electronic keyer, calibration and other functions. Overwhelmed by world-wide demand for its products, the fledgling indigenous Indian manufacturer (hfsignals.com) is no longer able to market the direct digital frequency synthesizer board ("Raduino") itself – leaving hundreds to thousands of customers world-wide hopeful of alternatives.

The Si5351 is an impressive phase locked loop frequency generator, typically based on a 25.000 MHz crystal. It comes in several flavors. The one utilized in the bitx series and in this application is capable of generating up to three frequencies from single-digit kHZ to past 160 MHz – and instantly upon command.

This provides a single-board frequency generator with impressive accuracy for testing or for receiver / transmitter development. Using simple code programmed into a free Integrated Development Environment, an application can be easily written (using free and available libraries) to allow easy control of the output frequencies either by computer or simple 10K ohm potentiometer to an Arduino input. Because the chip can generate three frequencies, just about every oscillator needed to create a superheterodyne receiver/transmitter can be built with this circuit alone.

Indeed, a thriving volunteer community around the world has been developing free third-party replacement software that can easily be loaded into this single board system, and/or into the BitX series of low-power transceivers.

This application note utilizes an available Adafruit Si5351 breakout board, on which the SMD soldering has already been accomplished, so that the board can be installed by simple pin-soldering or using a simple header socket and soldered pins. The Adafruit Si5351 breakout board is available directly from Adafruit or through Amazon or other sources.

Extremely accurate calibration of the 25.000 MHz crystal standard can be accomplished by commanding the system to generate exactly that frequency (25.000 MHz) and then measuring it on an accurate receiver --- or by commanding the generation of a frequency of a readily-receivable WWV frequency standard station, beating the two signals against each other at the input to any receiver, and making the appropriate correction to the 25 000 000 static figure utilized in the code for the frequency of the crystal standard. Once this is accomplished, all three clock outputs will be corrected for all frequencies commanded, to the degree that the error has been corrected. There may still be temperature effects, as the crystal on the Adafruit board is not oven-stablized in this design. Accuracy to within 100 Hz in the HF band should be easily achievable.

This signal source can be utilized to replace drifting oscillators in older radio designs. The Arduino code of many volunteer developments includes the ability to follow commands issued by computer control presuming a FT-817 transceiver --- making computer control of older transceivers possible.

The output signal of the Si5351 chip is a dc-coupled CMOS square wave (with attendant harmonics) of appxoimately 3 volts peak-to-peak magnitude when lightly loaded (e.g., 10kohms). With lower impedance loads (e.g. 50 ohms) the output magnitude will be lower. The DC bias (approximately 1.5 V) may be removed with a series capacitor (e.g. .001 or .01 microfarads depending on load impedance). The signal may also be amplified or buffered by any number of transistor amplifiers.

**ERRATA: Pin 5 of the digital 16x2 display must be grounded. The original printed circuit left it floating. Run a jumper from pin 5 to pin 1 or any other easily accessible ground point.**

**WARNINGS:**

- Do not apply more than 12V to the power input as the voltage regulator may get too hot.
- Do not apply more than 5V to the A7 FREQ input pin, or any other input pin of the Arduino.

- The +5VDC near A7 is intended to supply the potentiometer connected to A7; do not short this unprotected power busline to ground.
- The 3 clock outputs from the Si5351 are CMOS outputs and are DC coupled. You may wish to add a series capacitor (e.g. .001 uf) to remove the DC bias of approximately 1.5 volts from this square wave signal.

## INPUTS

| | | |
|---|---|---|
| RESET | | not used |
| A0, A6 | | used by the SWR forward/reverse measurement circuitry |
| A1 Key | | for morse code key |
| A2 Cal | | older version of calibrate function |
| A3 FCN | | Function switch (used in more recent bitx40 sketches for most adjustments) |
| A4, A5 | | Not available – used internally to control the Si5351 via I2C interface |
| A7 | | potentiometer wiper input 0-5V to control frequency |

## OUTPUTS

| | | |
|---|---|---|
| D2 | | digital output unused |
| D3, D5, D6 | | pulse-width modulated pseudo analog outputs available for your usage |
| D4, D7 | | apparently unused |
| D8,D9,D10,D11,D12,D13 | used to control the display | D8 → RW<br>D9 →E<br>D10--> |

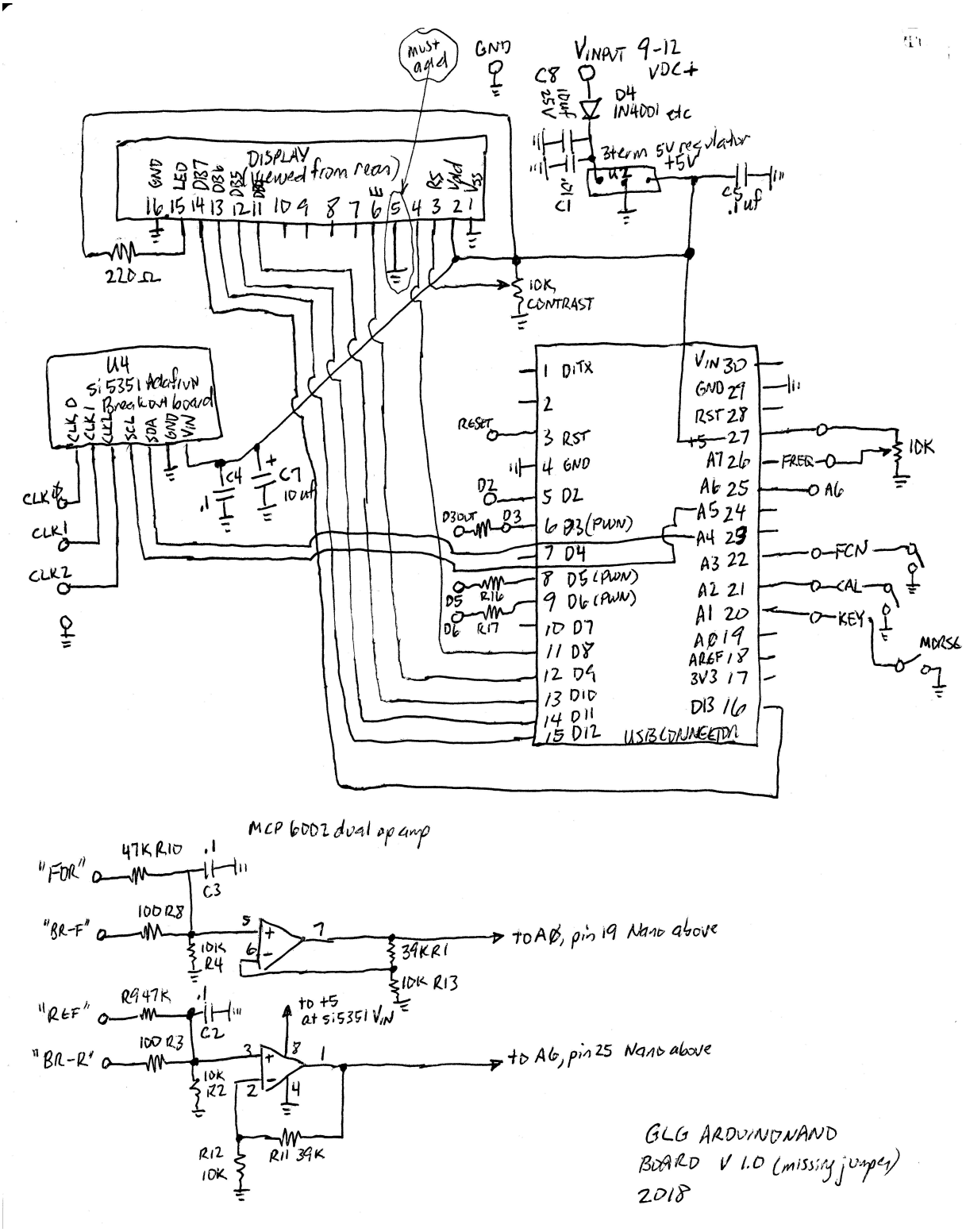
## COMPONENTS & PURPOSE

| Component | Function | Notes |
|---|---|---|
| Arduino Nano | microprocessor with analog / digital inputs & output | Strongly suggest using “header” sockets to allow plugging in the Nano rather than soldering it in. |

| | | |
|---|---|---|
| Adafruit Si5351 module | Digital frequency synthesis, includes its own onboard crystal | https://learn.adafruit.com/adafruit-si5351-clock-generator-breakout/overview<br><br>https://www.amazon.com/Adafruit-Si5351A-Clock-Generator-Breakout/dp/B00SK8LNYG |
| 16 character x 2 lines parallel connection LCD display (not the I2C ported version | Display – available in many different cloned versions on eBay | |
| U2 3 terminal 5-volt regulator | Provides regulated +5V DC for the display, the si5351 chip and the Arduino Nano | Does not normally require a heat sink. |
| D4 | Any 1Amp silicon rectifier, such as 1N4001, 1N4004, 1N4007 | Protects circuitry against accidental reverse polarity from the 9-12V input power |
| R6 10K | Adjust display contrast | miniature trimmer potentiometer |
| R14 200 ohms | limits current draw by display, limits brightness | |
| C1, C4, C5 0.1 uf | ceramic capacitor for bypass to filter out AC from DC supply lines | |
| C7, C8, 10-40 uf / 25V | filter capacitors for DC lines | |
| | | |
| REMAINING COMPONENTS FOR MEASURING SWR BRIDGE OUTPUT VOLTAGES AND CONTROLLING PANEL METER | | |
| U5 | | |
| R5, R16, R17 | PWM output pins | Select values to give proper full scale on measuring scale |
| | | |
| REMAINING COMPONENTS RELATED TO SWR METER MEASUREMENTS | | |
| MCP6002 dual op amp | Op amp gain set to 4. Input dividers may be used to determine total gain. Arduino can measure voltages from 0-5 | this opamp should be socketed, and will protect the Arduino from out of range input voltages |

| | | |
|---|---|---|
| | V. Two input impedances are provided, 10K and 57K | |
| | | |
| | | |
| | | |

## SCHEMATIC OF BOARD



# PINOUT DIFFERENCES BETWEEN ARDUINO NANO AND ARDUINO UNO APPLICATIONS

| Signal to the 16x2 display | Arduino Nano in hfsignals bitx40 Raduino and on the KX4Z printed circuit Bord | Arduino UNO Code on Elegoo Experimenter's kit |
|---|---|---|
| rs | 8 | 12 |
| en | 9 | 11 |
| d4 | 10 | 5 |
| d5 | 11 | 4 |
| d6 | 12 | 3 |
| d7 | 13 | 2 |

| Signal | bitx40 Raduino | KX4Z board | ELEGOO UNO BOARD |
|---|---|---|---|
| SDA | A4 | A4 | A0 |
| SCL | A5 | A5 | A1 |
| | | | |

# EXAMPLE SOURCE CODE FOR ARDUINO INTEGRATED DEVELOPMENT ENVIRONMENT

```
 // include the library code:
 #include <LiquidCrystal.h>
 /**
   The Wire.h library is used to talk to the Si5351 and we also declare an instance of
   Si5351 object to control the clocks.
*/
#include <Wire.h>

#define BB0(x) ((uint8_t)x)           // Bust int32 into Bytes
#define BB1(x) ((uint8_t)(x>>8))
#define BB2(x) ((uint8_t)(x>>16))

#define SI5351BX_ADDR 0x60            // I2C address of Si5351   (typical)
#define SI5351BX_XTALPF 2             // 1:6pf  2:8pf  3:10pf

// If using 27mhz crystal, set XTAL=27000000, MSA=33.  Then vco=891mhz
#define SI5351BX_XTAL 25000000        // Crystal freq in Hz  breadpan version 460
#define SI5351BX_MSA  35              // VCOA is at 25mhz*35 = 875mhz

// User program may have reason to poke new values into these 3 RAM variables
uint32_t si5351bx_vcoa = (SI5351BX_XTAL*SI5351BX_MSA);  // 25mhzXtal calibrate
uint8_t  si5351bx_rdiv = 0;           // 0-7, CLK pin sees fout/(2**rdiv)
uint8_t  si5351bx_drive[3] = {3, 3, 3}; // 0=2ma 1=4ma 2=6ma 3=8ma for CLK 0,1,2

uint8_t  si5351bx_clken = 0xFF;       // Private, all CLK output drivers off

 // initialize the library by associating any needed LCD interface pin
 // with the arduino pin number it is connected to

 // These numbers for for the ADUINO UNO:
 const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
 char  string[16];
 char  buf1[6], buf2[5], buf3[10];
 //  use buf3 to hold the frequency for printing on the screen


 //
 // These numbers are for the ARDUINO NANO in the RADUINO:
 // for the NANO
 LiquidCrystal lcd(8, 9, 10, 11, 12, 13);
 //
```

```
 // These numbers are for the ARDUINO UNO in the Elegoo:
 //
//  LiquidCrystal lcd(rs, en, d4, d5, d6, d7);


  //For the UNO use A0 and A1
  //For the Nano A0 and A6



  // at least on the bridge I screwd these backwards.
  const int forwardpin = 6;
  const int reversepin = 0;

  // For the UNO we will use D9 because D3 is in use driving the screen....
  // For the NANO we will use D3 because D9 is in use driving the screen....
  const int outputpin = 6;

  // Potentiometer frequency control
  // for the UNO we are going to use A2
  // For the Nano we will use A7 just like the Bitx40
  const int freqpin = 7;


  long  frequency;  // parameter to send to si5351
  float ffrequency;  // float version of frequency in Hz
  void setup()
  {
    // In this sketch, we'll use the Arduino's serial port
    // to send text back to the main computer. For both sides to
    // communicate properly, they need to be set to the same speed.
    // We use the Serial.begin() function to initialize the port
    // and set the communications speed.

    // The speed is measured in bits per second, also known as
    // "baud rate". 9600 is a very commonly used baud rate,
    // and will transfer about 10 characters per second.

    Serial.begin(9600);
    pinMode(outputpin, OUTPUT);
    lcd.begin(16, 2);
    // Print a message to the LCD.
    // Set the cursor to 0th column, 0th row:
      lcd.setCursor(0, 0);
    // print the number of seconds since reset:
    //lcd.print(millis() / 1000);
```

```
  lcd.print("GLG SWR Meter");
  delay(1000);
  //initialize the SI5351
  si5351bx_init();
  Serial.println("*Initialized Si5351\n");
  lcd.setCursor(0,0);
  lcd.print("Si5351 on");
  si5351bx_setfreq(2, 25000000L);
  Serial.println("*Si5350 ON\n");
  delay(1000);
  frequency = 7000000L;

  lcd.setCursor(0,1);
  lcd.print("For/ref volt");
  delay(1000);


}


void loop()
{
  // We'll declare three floating-point variables
  // (We can declare multiple variables of the same type on one line:)

  float fvoltage, rvoltage, ratio,potvoltage;



  // First we'll measure the voltage at the analog pin. Normally
  // we'd use analogRead(), which returns a number from 0 to 1023.
  // Here we've written a function (further down) called
  // getVoltage() that returns the true voltage (0 to 5 Volts)
  // present on an analog input pin.

  potvoltage = getVoltage(freqpin);\
  Serial.print("Pot Wiper voltage =");
  Serial.print(potvoltage);
  Serial.println( " ");


  frequency = (long)  (potvoltage * 3000000.0);  // index to 5V = 15MHz
  ffrequency = (float) frequency;
  if (frequency<100000) frequency=100000L; // don't let it go too low....
  Serial.print("Frequency: ");
  Serial.print(frequency);
  Serial.println(" ");
```

```
 si5351bx_setfreq(2, frequency);

fvoltage = getVoltage(forwardpin);
rvoltage = getVoltage(reversepin);



//Serial.print("SWR = ");
//Serial.print(ratio);
//Serial.println(" ");
//Serial.println(" ");

// Now we'll use the serial port to print these values
// to the serial monitor!

// To open the serial monitor window, upload your code,
// then click the "magnifying glass" button at the right edge
// of the Arduino IDE toolbar. The serial monitor window
// will open.

// (NOTE: remember we said that the communication speed
// must be the same on both sides. Ensure that the baud rate
// control at the bottom of the window is set to 9600. If it
// isn't, change it to 9600.)


// To send data from the Arduino to the serial monitor window,-
// we use the Serial.print() function. You can print variables
// or text (within quotes).
//
// calculate the ratio of reflected voltage to forward voltage,


if (fvoltage>0) {

  if( (fvoltage-rvoltage)!= 0) {
  ratio =  (fvoltage+rvoltage)/(fvoltage - rvoltage);
  }
  else  ratio = 10;

  if( ratio<0 ) ratio = -1 * ratio;
```

```
     if(ratio<5.0) {
          analogWrite(outputpin, (int) (  (ratio-1.0)* (255/4) )   ) ;
     }
    else analogWrite(outputpin,255);
    //  run it to the top if nosignal;

    }  // end of if forward voltage non zero
    // otherwise no power.

    Serial.print("forwardvoltage: ");
    Serial.print(fvoltage);
    Serial.print("reversevoltage:  ");
    Serial.print(rvoltage);
    Serial.println(" ");
    Serial.print("SWR = ");
    Serial.print(ratio);
    Serial.println(" ");

    //fvoltage=1.2;
    //rvoltage=0.6;



//    dtostrf(fvoltage,4,2,buf1);
    dtostrf(rvoltage,4,2,buf2);
    dtostrf( ((ffrequency)/1000000.0),5,1,buf1);
    dtostrf(  ratio,4,2,buf2);
    sprintf(string,"Fq:%s SWR:%s",buf1,buf2);

    //    sprintf(string,"F:%s R:%s",buf1,buf2);
    lcd.setCursor(0,1);
    lcd.print(string);
    lcd.setCursor(0,0);
    sprintf(string,"Freq:%10ld ", (frequency/1000));
    lcd.print(string);




    // Note that all of the above statements are "print", except
    // for the last one, which is "println". "Print" will output
    // text to the SAME LINE, similar to building a sentence
    // out of words. "Println" will insert a "carriage return"
    // character at the end of whatever it prints, moving down
    // to the NEXT line.
```

```
  delay(1000); // repeat 10 per second (change as you wish!)
  Serial.println(frequency);


}


float getVoltage(int pin)
{
  // This function has one input parameter, the analog pin number
  // to read. You might notice that this function does not have
  // "void" in front of it; this is because it returns a floating-
  // point value, which is the true voltage on that pin (0 to 5V).

  // You can write your own functions that take in parameters
  // and return values. Here's how:

    // To take in parameters, put their type and name in the
    // parenthesis after the function name (see above). You can
    // have multiple parameters, separated with commas.

    // To return a value, put the type BEFORE the function name
    // (see "float", above), and use a return() statement in your code
    // to actually return the value (see below).

    // If you don't need to get any parameters, you can just put
    // "()" after the function name.

    // If you don't need to return a value, just write "void" before
    // the function name.

  // Here's the return statement for this function. We're doing
  // all the math we need to do within this statement:

  return (analogRead(pin) * 0.004882814);

  // This equation converts the 0 to 1023 value that analogRead()
  // returns, into a 0.0 to 5.0 value that is the true voltage
  // being read at that pin.
}

// Other things to try with this code:

//   Turn on an LED if the temperature is above or below a value.

//   Read that threshold value from a potentiometer - now you've
//   created a thermostat!
```

```
// *************  SI5315 routines - tks Jerry Gaffke, KE7ER   ***********************

// An minimalist standalone set of Si5351 routines.
// VCOA is fixed at 875mhz, VCOB not used.
// The output msynth dividers are used to generate 3 independent clocks
// with 1hz resolution to any frequency between 4khz and 109mhz.

// Usage:
// Call si5351bx_init() once at startup with no args;
// Call si5351bx_setfreq(clknum, freq) each time one of the
// three output CLK pins is to be updated to a new frequency.
// A freq of 0 serves to shut down that output clock.

// The global variable si5351bx_vcoa starts out equal to the nominal VCOA
// frequency of 25mhz*35 = 875000000 Hz.  To correct for 25mhz crystal errors,
// the user can adjust this value.  The vco frequency will not change but
// the number used for the (a+b/c) output msynth calculations is affected.
// Example:  We call for a 5mhz signal, but it measures to be 5.001mhz.
// So the actual vcoa frequency is 875mhz*5.001/5.000 = 875175000 Hz,
// To correct for this error:     si5351bx_vcoa=875175000;

// Most users will never need to generate clocks below 500khz.
// But it is possible to do so by loading a value between 0 and 7 into
// the global variable si5351bx_rdiv, be sure to return it to a value of 0
// before setting some other CLK output pin.  The affected clock will be
// divided down by a power of two defined by  2**si5351_rdiv
// A value of zero gives a divide factor of 1, a value of 7 divides by 128.
// This lightweight method is a reasonable compromise for a seldom used feature.

void si5351bx_init() {                 // Call once at power-up, start PLLA
  uint8_t reg;  uint32_t msxp1;
  Wire.begin();
  i2cWrite(149, 0);                    // SpreadSpectrum off
  i2cWrite(3, si5351bx_clken);         // Disable all CLK output drivers
  i2cWrite(183, SI5351BX_XTALPF << 6); // Set 25mhz crystal load capacitance
  msxp1 = 128 * SI5351BX_MSA - 512;   // and msxp2=0, msxp3=1, not fractional
  uint8_t  vals[8] = {0, 1, BB2(msxp1), BB1(msxp1), BB0(msxp1), 0, 0, 0};
  i2cWriten(26, vals, 8);              // Write to 8 PLLA msynth regs
  i2cWrite(177, 0x20);                 // Reset PLLA  (0x80 resets PLLB)
  // for (reg=16; reg<=23; reg++) i2cWrite(reg, 0x80);    // Powerdown CLK's
  // i2cWrite(187, 0);                 // No fannout of clkin, xtal, ms0, ms4
}

void si5351bx_setfreq(uint8_t clknum, uint32_t fout) {  // Set a CLK to fout Hz
  uint32_t  msa, msb, msc, msxp1, msxp2, msxp3p2top;
  if ((fout < 500000) || (fout > 109000000)) // If clock freq out of range 0.5MHz-109 MHz
    si5351bx_clken |= 1 << clknum;      //  shut down the clock
```

```
  else {
    msa = si5351bx_vcoa / fout;     // Integer part of vco/fout
    msb = si5351bx_vcoa % fout;     // Fractional part of vco/fout
    msc = fout;             // Divide by 2 till fits in reg
    while (msc & 0xfff00000) {
      msb = msb >> 1;
      msc = msc >> 1;
    }
    msxp1 = (128 * msa + 128 * msb / msc - 512) | (((uint32_t)si5351bx_rdiv) << 20);
    msxp2 = 128 * msb - 128 * msb / msc * msc; // msxp3 == msc;
    msxp3p2top = (((msc & 0x0F0000) << 4) | msxp2);     // 2 top nibbles
    uint8_t vals[8] = { BB1(msc), BB0(msc), BB2(msxp1), BB1(msxp1),
                        BB0(msxp1), BB2(msxp3p2top), BB1(msxp2), BB0(msxp2)
                      };
    i2cWriten(42 + (clknum * 8), vals, 8); // Write to 8 msynth regs
    i2cWrite(16 + clknum, 0x0C | si5351bx_drive[clknum]); // use local msynth
    si5351bx_clken &= ~(1 << clknum);   // Clear bit to enable clock
  }
  i2cWrite(3, si5351bx_clken);        // Enable/disable clock
}

void i2cWrite(uint8_t reg, uint8_t val) {   // write reg via i2c
  Wire.beginTransmission(SI5351BX_ADDR);
  Wire.write(reg);
  Wire.write(val);
  Wire.endTransmission();
}

void i2cWriten(uint8_t reg, uint8_t *vals, uint8_t vcnt) {  // write array
  Wire.beginTransmission(SI5351BX_ADDR);
  Wire.write(reg);
  while (vcnt--) Wire.write(*vals++);
  Wire.endTransmission();
}

// ********** End of Jerry's si5315bx routines
**********************************************************
```