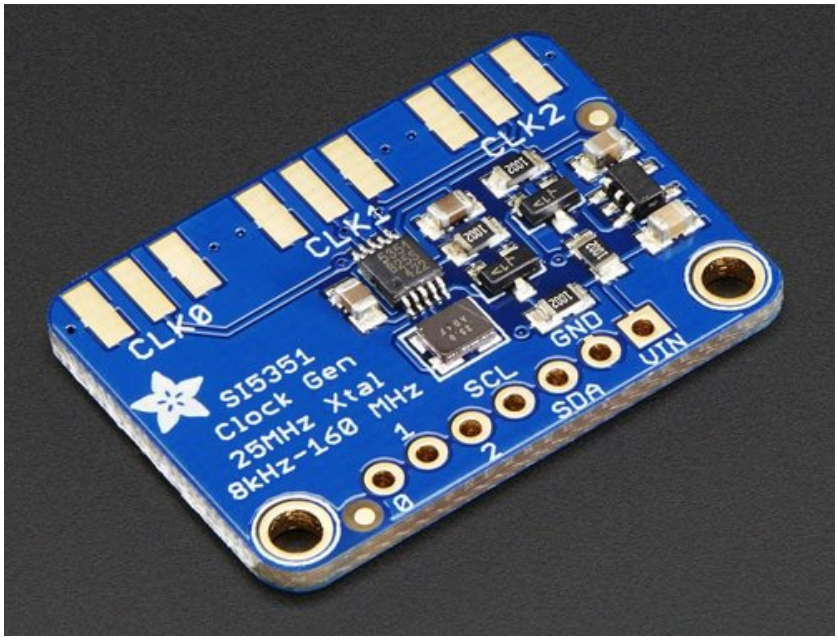
Adafruit Si5351 Clock Generator Breakout

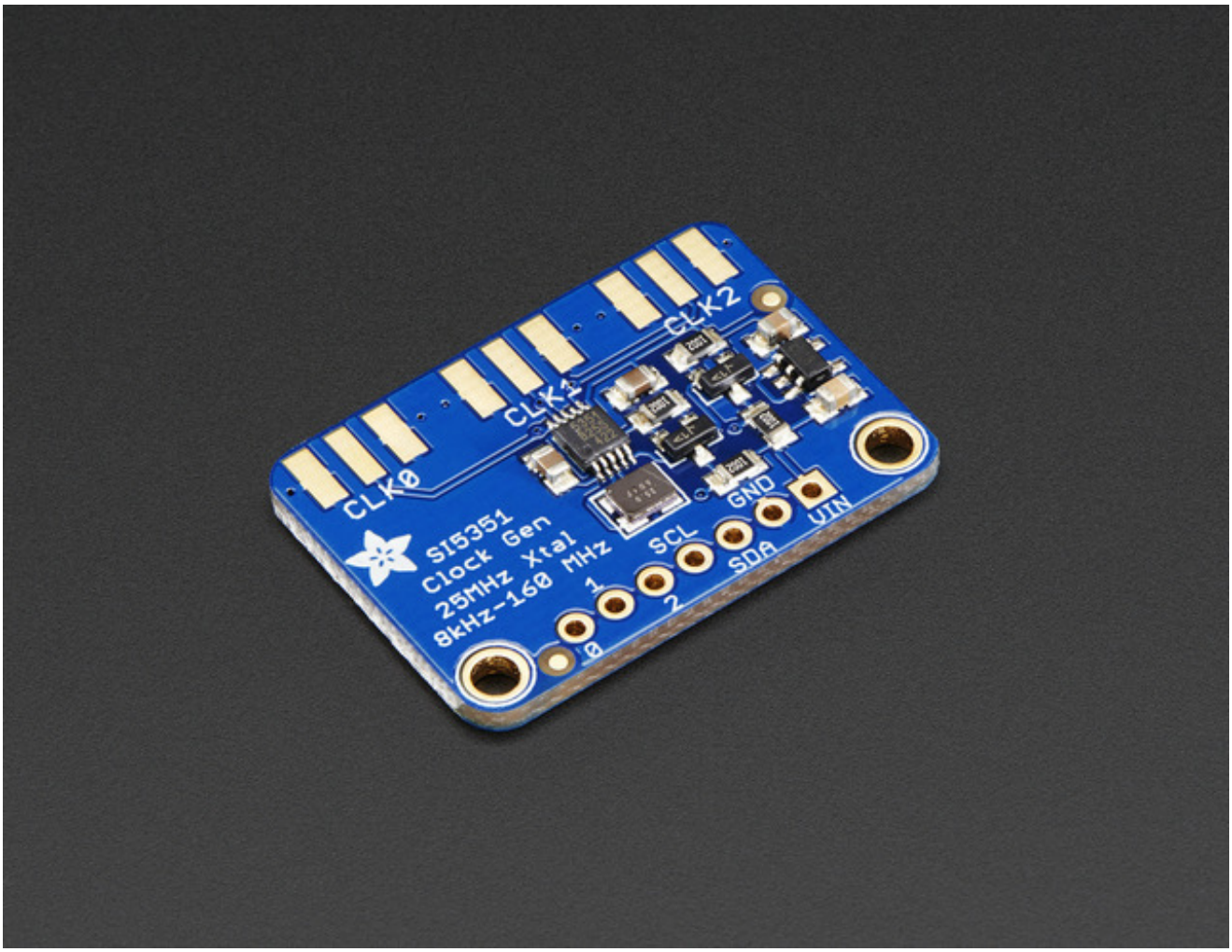
Created by lady ada



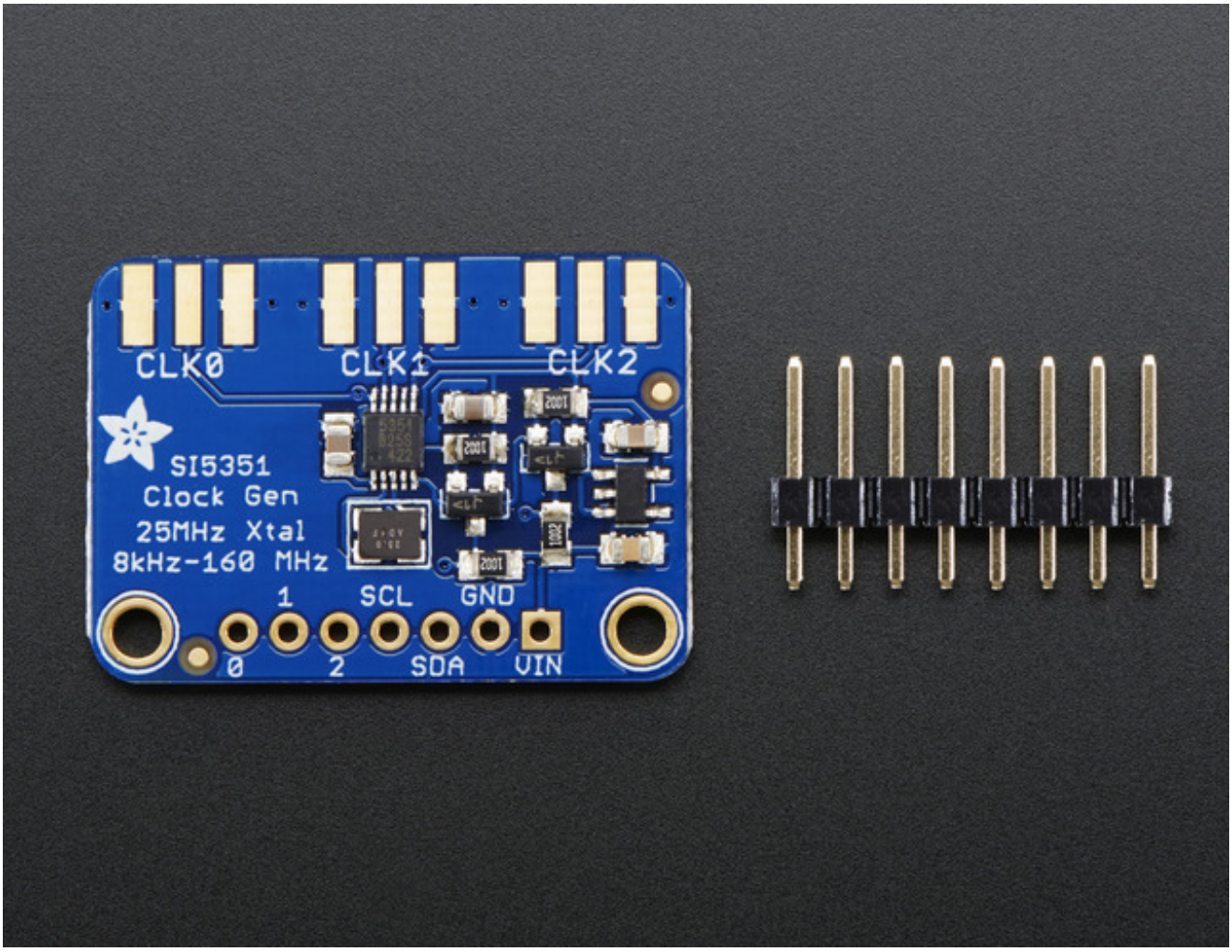
Last updated on 2016-08-04 11:04:22 PM UTC

Guide Contents

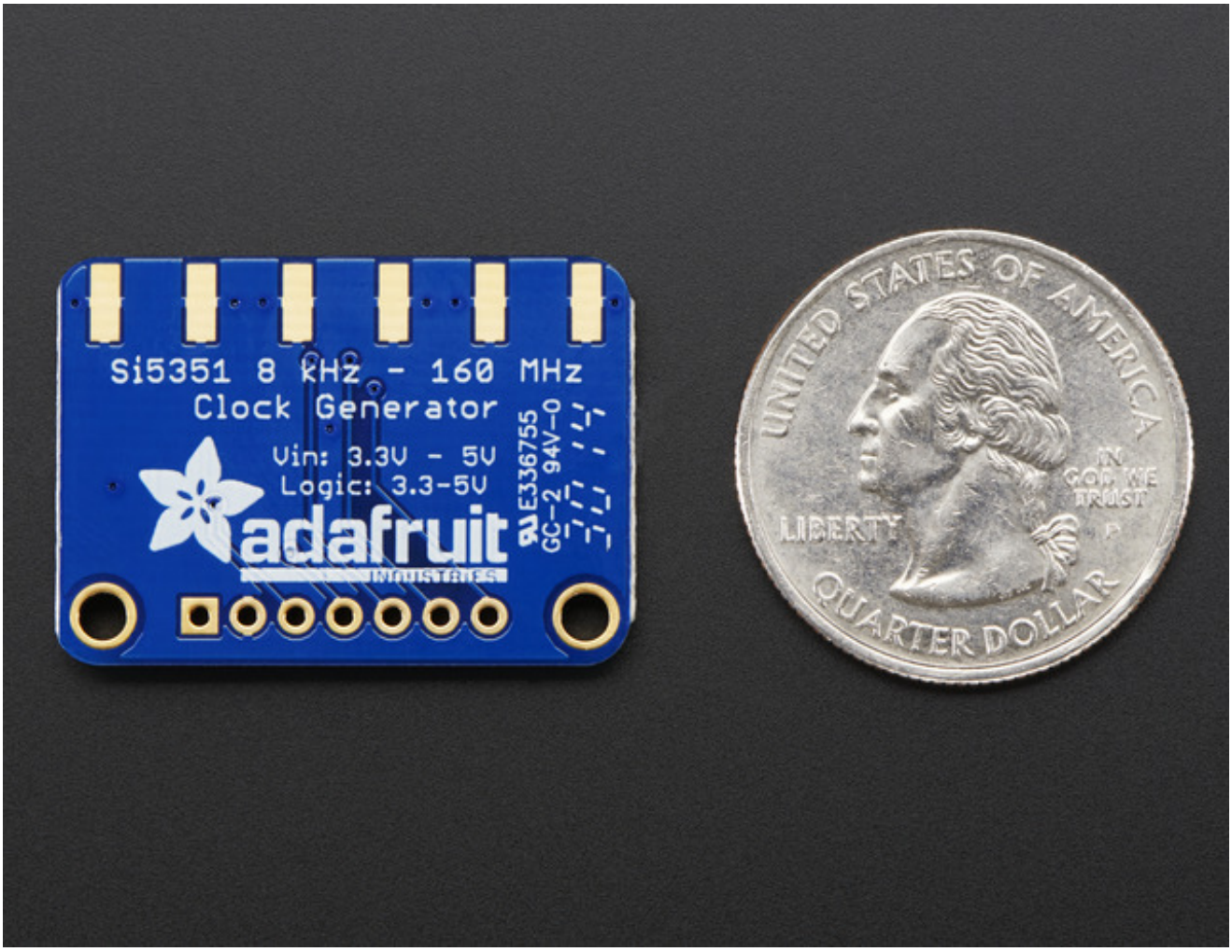
Overview



Never hunt around for another crystal again, with the Si5351 clock generator breakout from Adafruit! This chip has a precision 25MHz crystal reference and internal PLL and dividers so it can generate just about any frequency, from <8KHz up to 150+ MHz.



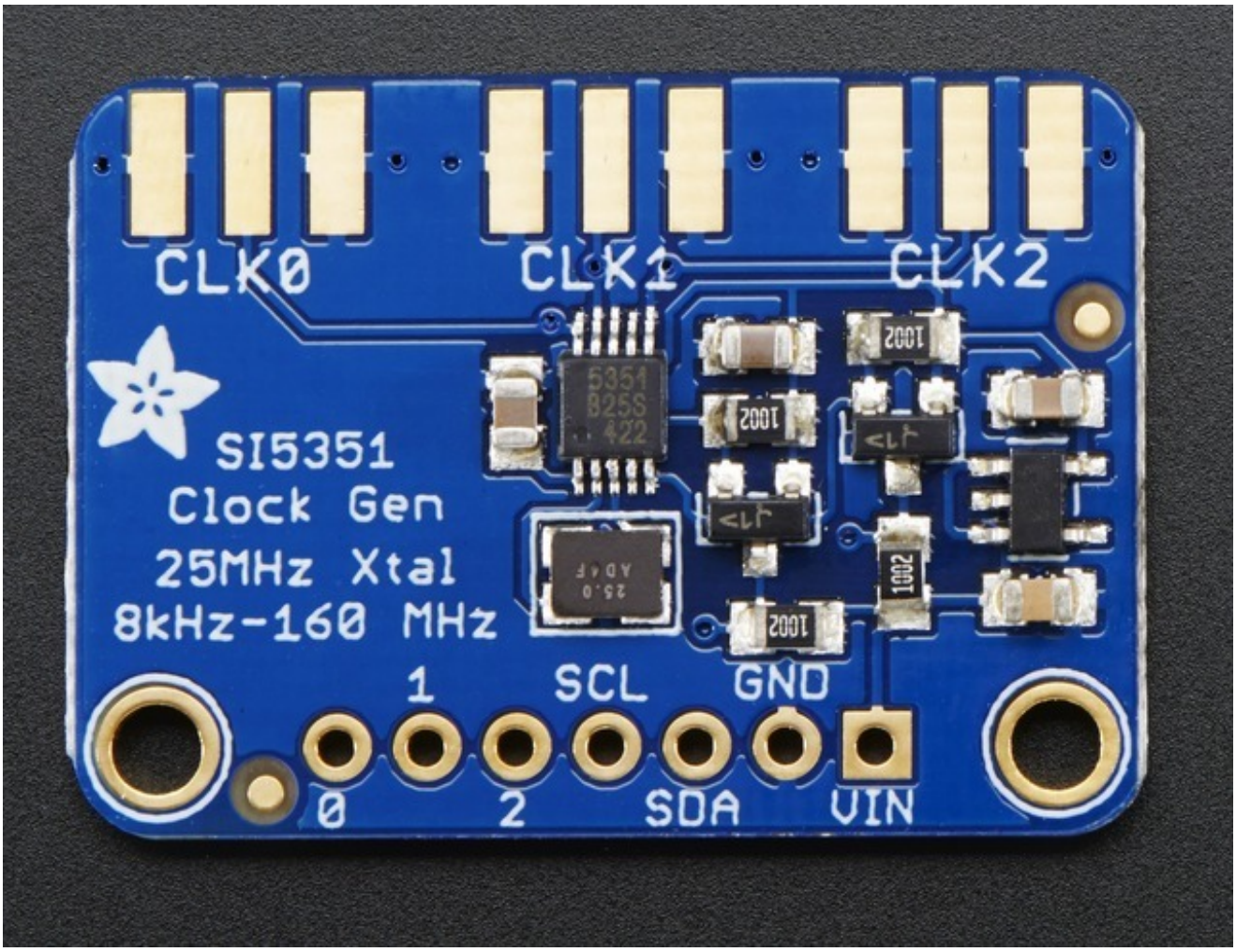
The Si5351 clock generator is an I2C controller clock generator. It uses the onboard precision clock to drive multiple PLL's and clock dividers using I2C instructions. By setting up the PLL and dividers you can create precise and arbitrary frequencies. There are three independent outputs, and each one can have a different frequency. Outputs are 3Vpp, either through a breadboard-friendly header or, for RF work, an optional SMA connector.



We put this handy little chip onto it's own breakout board PCB, with a 3.3V LDO regulator so it can be powered from 3-5VDC. We also put level shifting circuitry on the I2C lines so you can use this chip safely with 3V or 5V logic.

Best of all, we even have a great tutorial and library to get you started! Our code is designed for use with the Arduino microcontroller and IDE but is easily ported to your favorite platform with I2C support

Pinouts



Power Pins

The clock generator on the breakout requires 3V power. Since many customers have 5V microcontrollers like Arduino, we tossed a 3.3V regulator on the board. Its ultra-low dropout so you can power it from 3.3V-5V just fine.

- **Vin** - this is the power pin. Since the chip uses 3 VDC, we have included a voltage regulator on board that will take 3-5VDC and safely convert it down. To power the board, give it the same power as the logic level of your microcontroller - e.g. for a 5V micro like Arduino, use 5V
- **GND** - common ground for power and logic

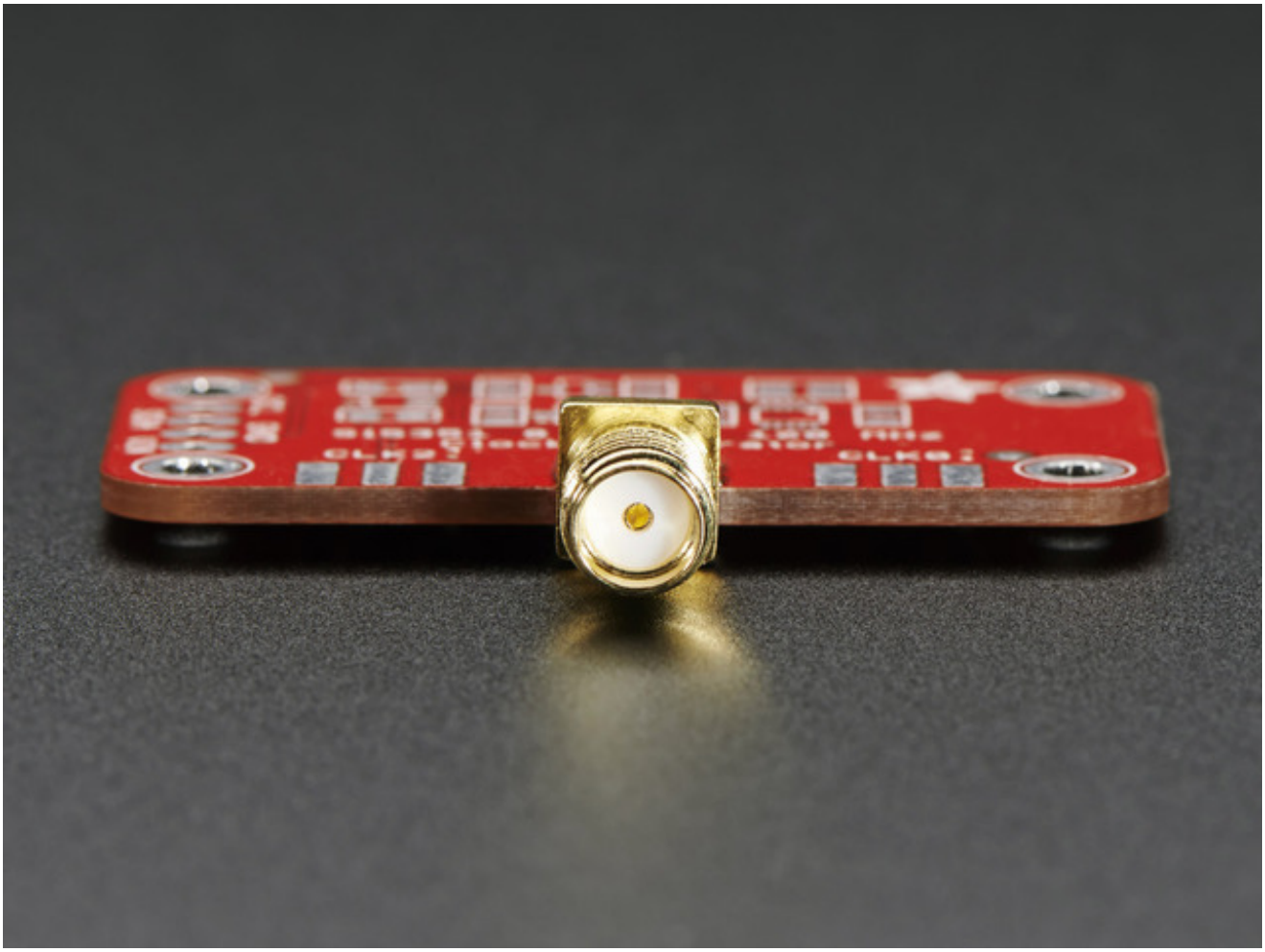
I2C Pins

- **SCL** - I2C clock pin, connect to your microcontrollers I2C clock line. This pin is level shifted so you can use 3-5V logic, and there's a 10K pullup on this pin.
- **SDA** - I2C data pin, connect to your microcontrollers I2C data line. This pin is level shifted so you can use 3-5V logic, and there's a 10K pullup on this pin.

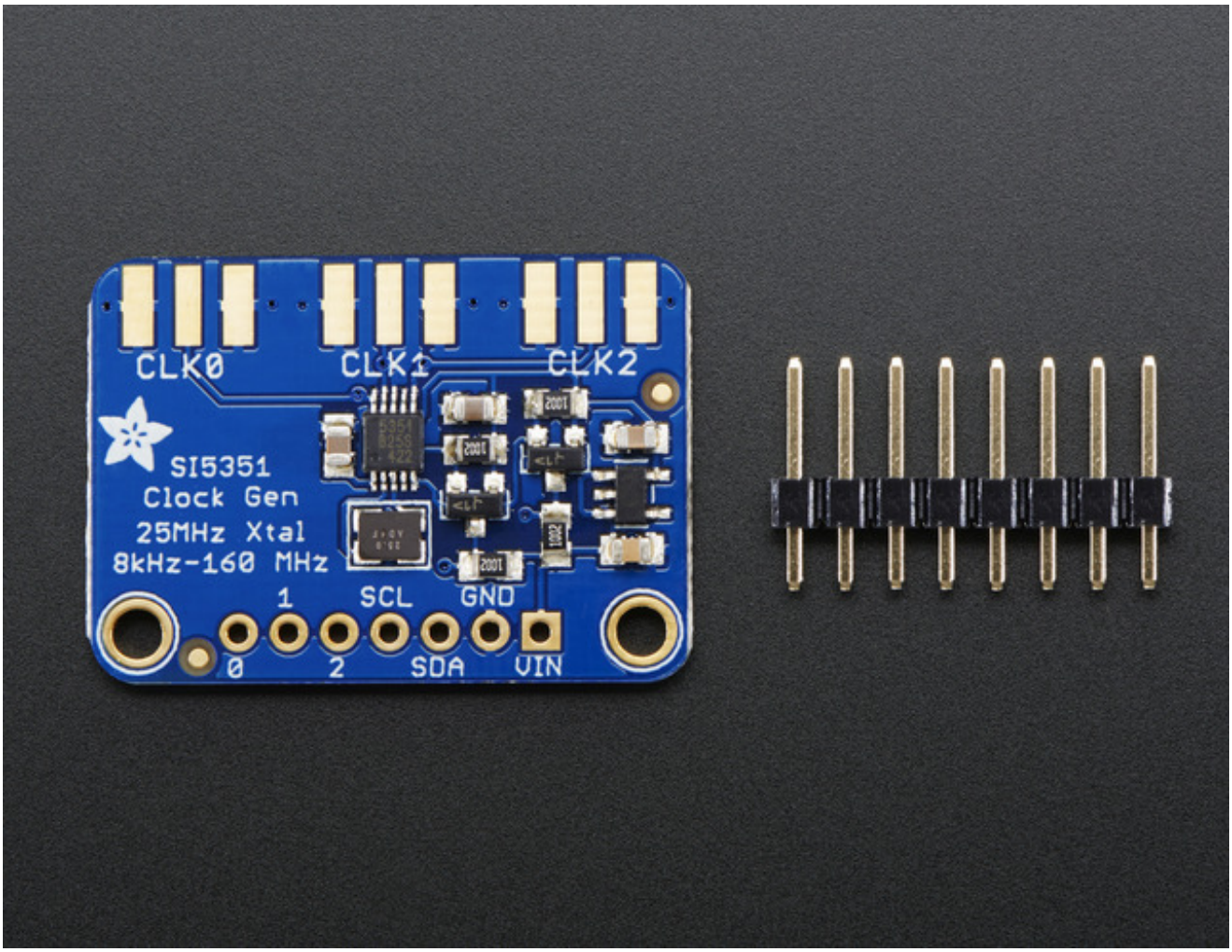
Clock Out Pins

- **0**, **1**, and **2** - These are the 3 independent clock generated outputs. They are square waves, from 0-3V.

The clock out pins are also brought out to SMA edge-launch connectors on the other side of the PCB. [You can purchase and solder on some edge-launch SMA connectors](http://adafru.it/dPr) if you want to pipe the signal into an RF cable.

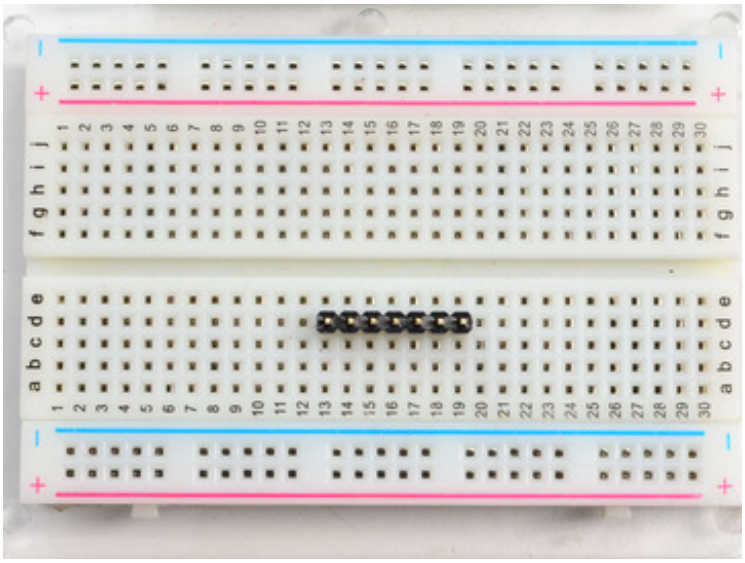


Assembly



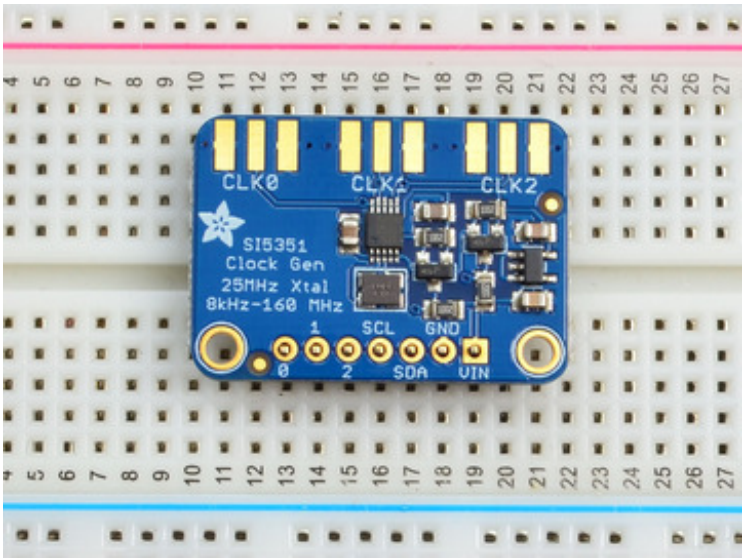
If you have the breadboard version of this sensor, you'll want to solder some header onto the sensor so it can be used in a breadboard.

Prepare the header



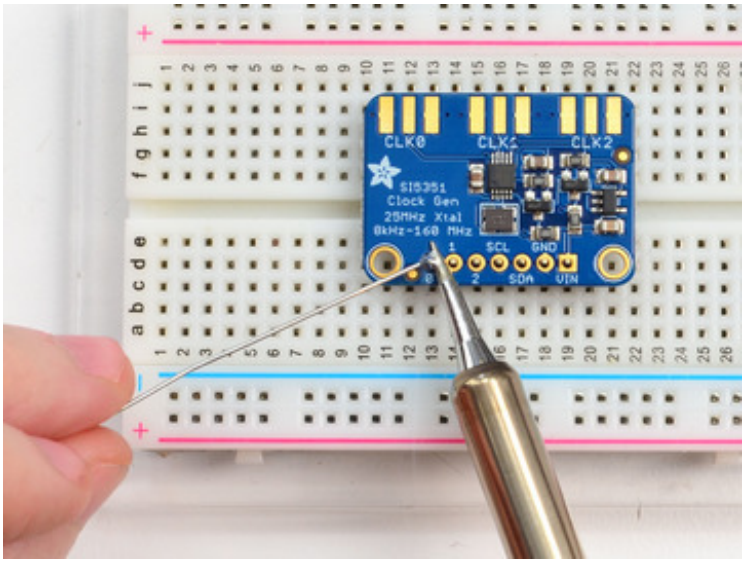
strip:

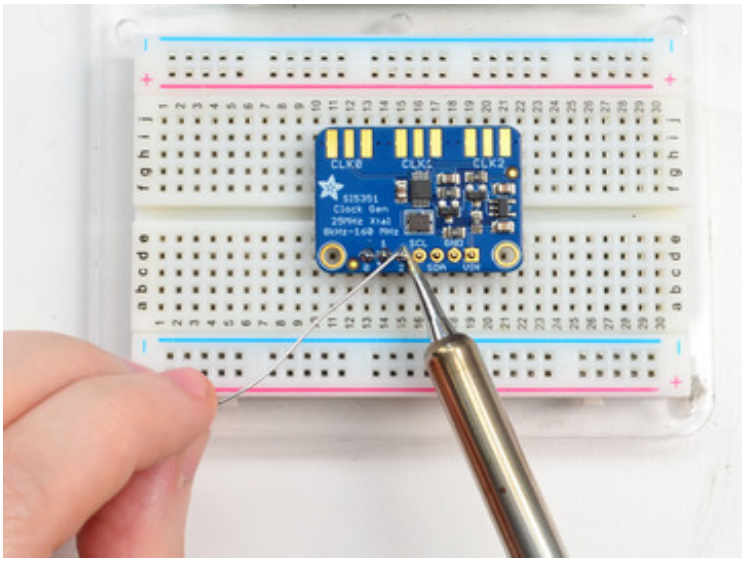
Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**

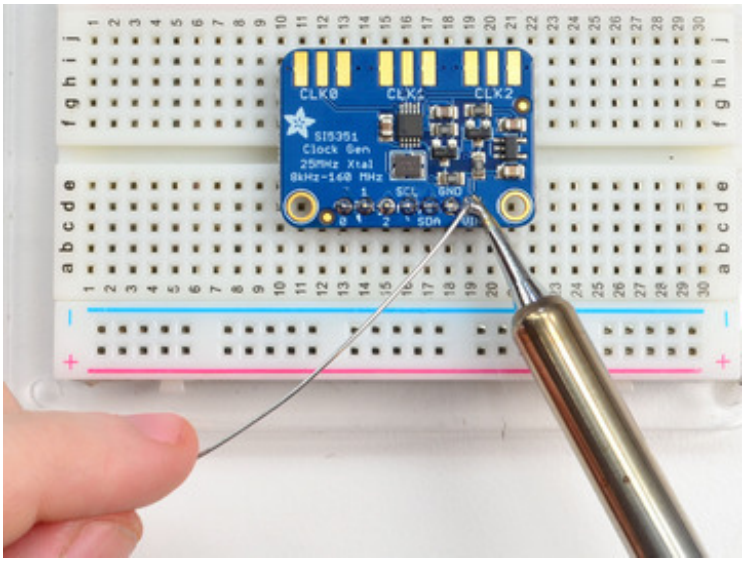


Add the breakout board:

Place the breakout board over the pins so that the short pins poke through the breakout pads

- 

- 

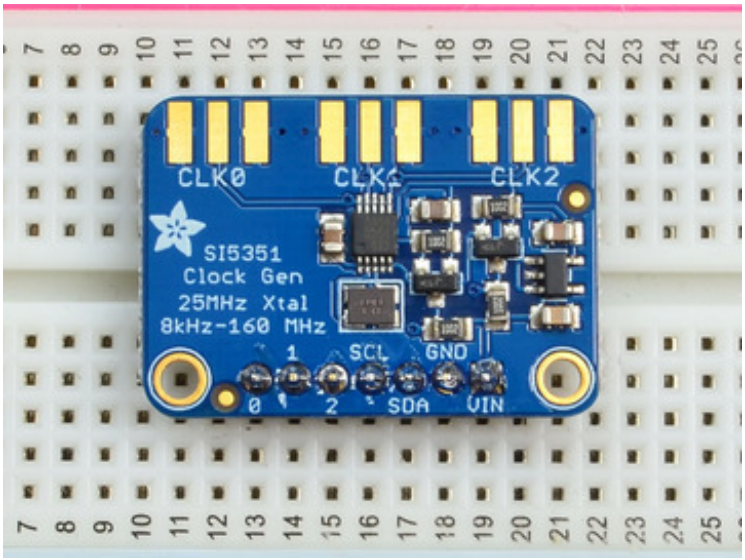
- 

And Solder!

Be sure to solder all pins for reliable electrical contact.

Solder the longer power/data strip first

(For tips on soldering, be sure to check out our [Guide to Excellent Soldering](http://adafru.it/aTk) (http://adafru.it/aTk).

- 

You're done! Check your solder joints visually and continue onto the next steps

Wiring & Test

Wiring for Arduino

You can easily wire this breakout to any microcontroller, we'll be using an Arduino. For another kind of microcontroller, just make sure it has I2C capability, then port the code - its pretty simple stuff!

(http://adafru.it/dPs)

(http://adafru.it/pBC)

- Connect **Vin** to the power supply, 3-5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V
- Connect **GND** to common power/data ground
- Connect the **SCL** pin to the I2C clock**SCL** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A5**, on a Mega it is also known as **digital 21** and on a Leonardo/Micro, **digital 3**
- Connect the **SDA** pin to the I2C data**SDA** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A4**, on a Mega it is also known as **digital 20** and on a Leonardo/Micro, **digital 2**

Download Adafruit_Si5351

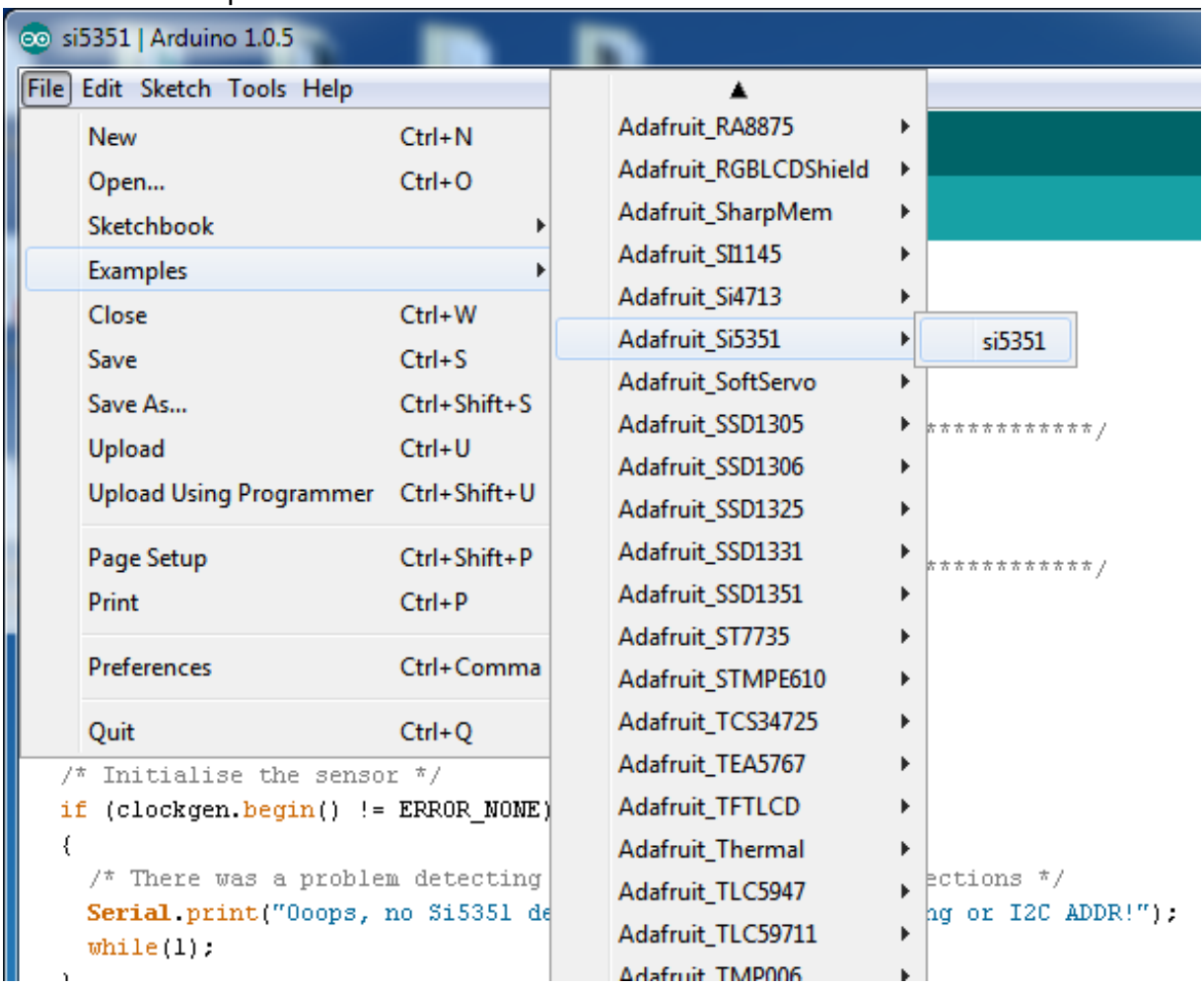
To begin reading sensor data, you will need todownload the Adafruit_Si5351 Library from our github repository (http://adafru.it/dPu). You can do that by visiting the github repo and manually downloading or, easier, just click this button to download the zip
Download the Adafruit Si5351 Library
http://adafru.it/dPu
Rename the uncompressed folder **Adafruit_Si5351** and check that the **Adafruit_Si5351** folder contains **Adafruit_Si5351.cpp** and **Adafruit_Si5351.h**

Place the **Adafruit_Si5351** library folder your **arduinosketchfolder/libraries/** folder.
You may need to create the**libraries** subfolder if its your first library. Restart the IDE.

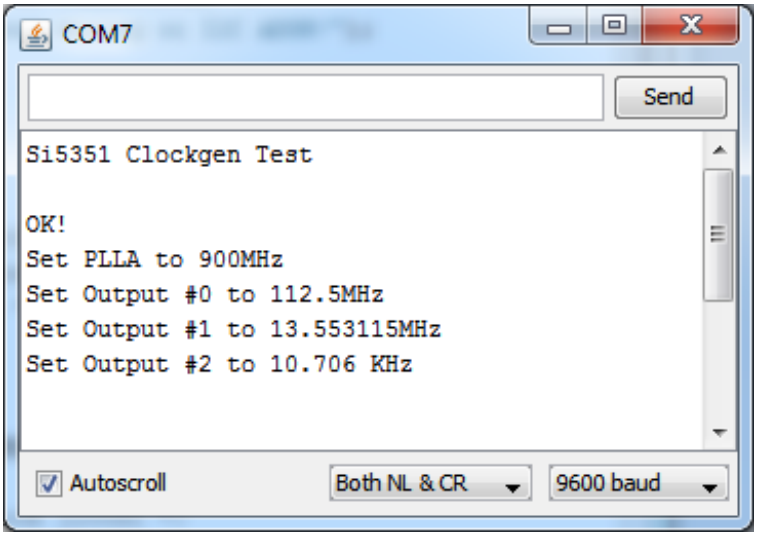
We also have a great tutorial on Arduino library installation at:
http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use (http://adafru.it/aYM)

Load Demo Sketch

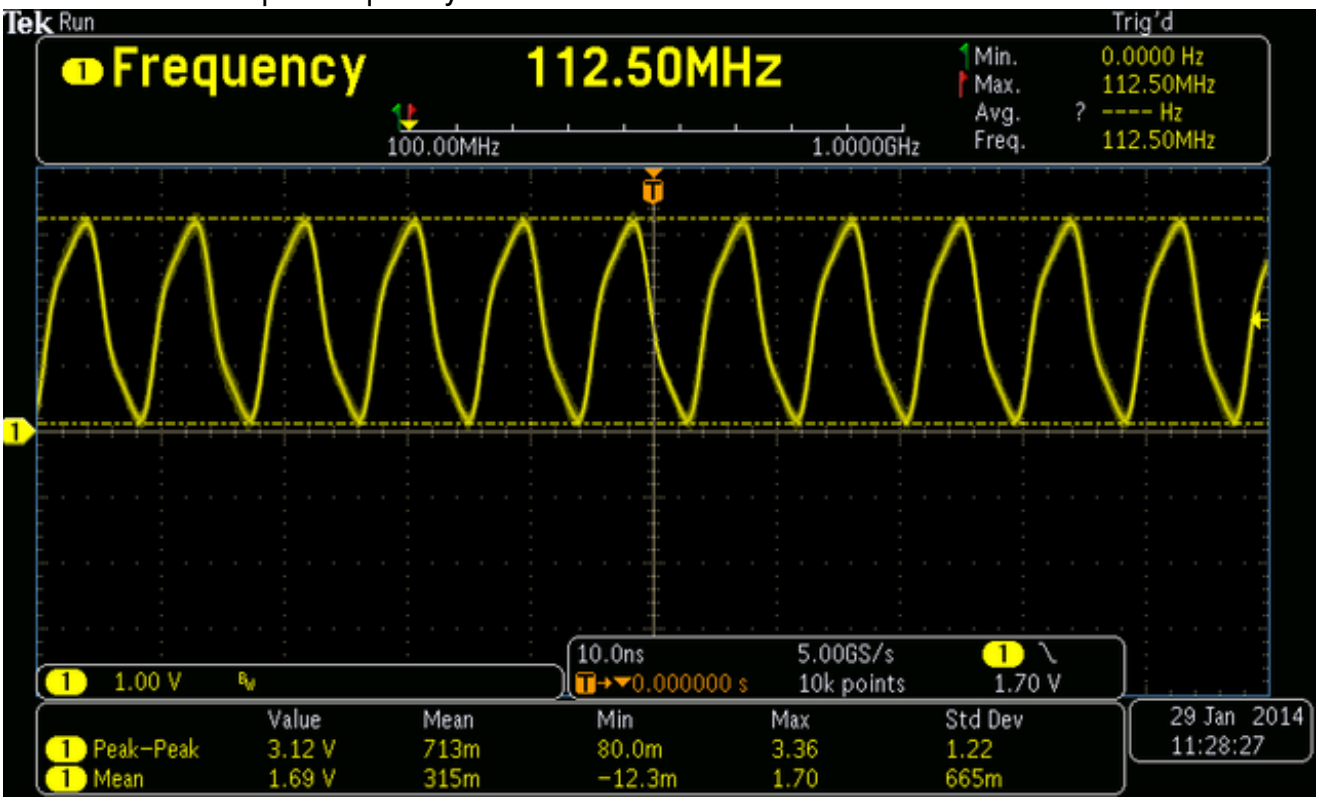
Now you can open up **File->Examples->Adafruit_Si5351->Si5351** and upload to your Arduino wired up to the sensor

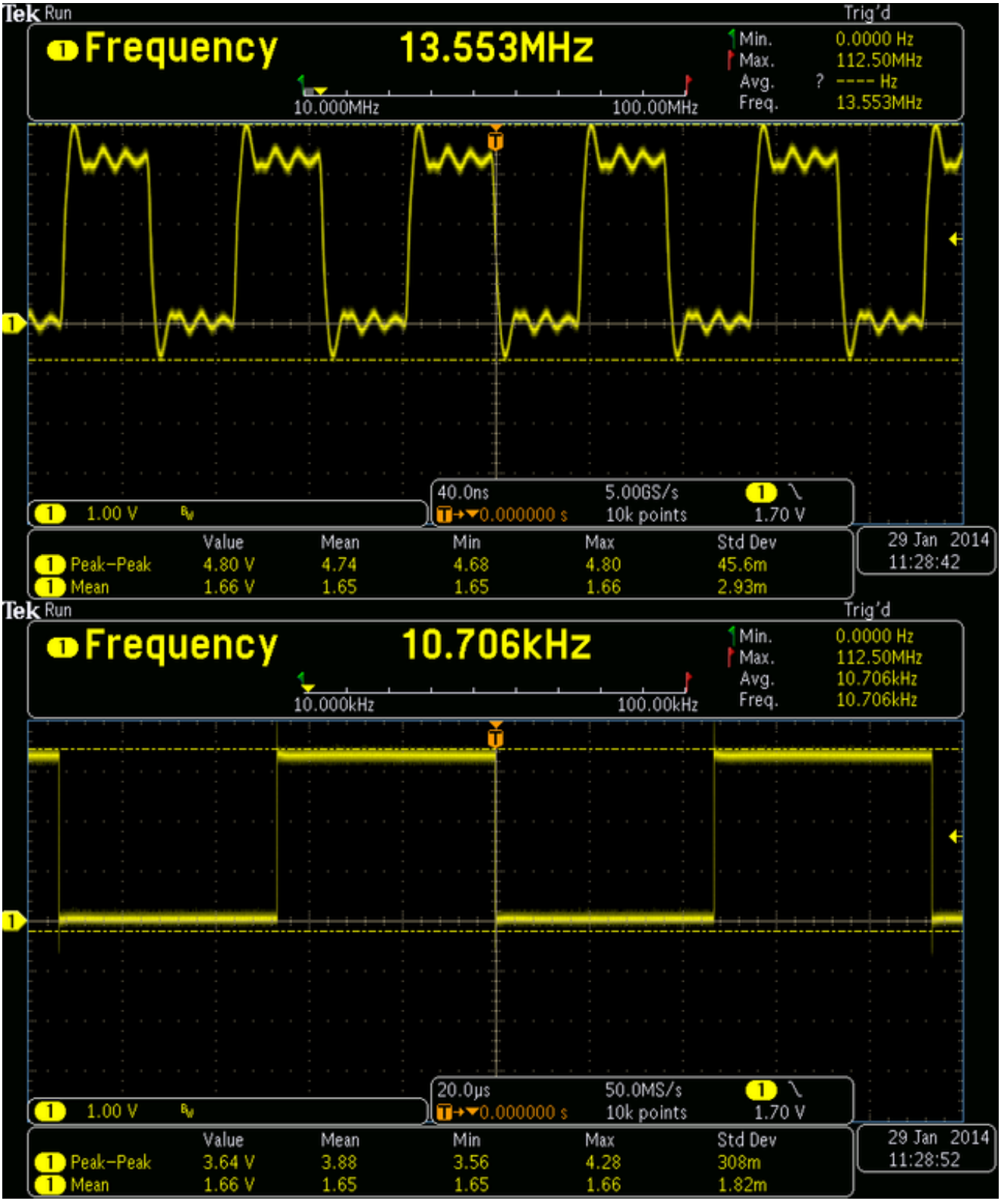


Then open up the Serial console at 9600 baud to check the output. You should see the following:



Now you can use your oscilloscope to probe the #0, #1 and #2 outputs
Depending on your oscillioscope make and model, it may not be possible for you to verify the 112.5MHz output frequency!





That's it! If you want to change the frequencies, adjust the example sketch and re-upload.

Library Reference

The library we have is simple and easy to use

You can create the **Adafruit_Si5351** object with:

```
Adafruit_SI5351 clockgen = Adafruit_SI5351();
```

I2C does not have pins, as they are fixed in hardware.

Begin!

To initialize the chip, call **clockgen.begin()** which will check that it can be found. Begin() returns true/false depending on these checks. We suggest you wrap begin() in a statement that will check if the chip was located:

```
if (clockgen.begin() != ERROR_NONE)
{
  /* There was a problem detecting the IC ... check your connections */
  Serial.print("Ooops, no Si5351 detected ... Check your wiring or I2C ADDR!");
  while(1);
}
```

Set up the PLL

The chip uses two subsections to generate clock outputs. First it **multiplies** the 25MHz reference clock by some amount (setting up the PLL), then it **divides** that new clock by some other amount (setting up the clock divider)

By noodling with the multiplier and divider you can generate just about any clock frequency!

There are **two** PLL multipliers (A and B), so if you want to have three outputs, two outputs will have to share one PLL.

Set up the PLL with 'integer mode'

The cleanest way to run the PLL is to do a straight up integer multiplication:

```
clockgen.setupPLLInt(SI5351_PLL_A or SI5351_PLL_B, m);
```

This sets **PLL_A** or **PLL_B** to be **25MHz * m** and **m** (the integer multipler) can range from **15** to **90**!

Set up the PLL with 'fractional mode'

This mode allows a much more flexible PLL setting by using fractional multipliers for the PLL setup, however, the output may have a slight amount of jitter so if possible, try to use integer mode!

clockgen.setupPLLInt(SI5351_PLL_A or SI5351_PLL_B, m, n, d);

This sets **PLL_A** or **PLL_B** to be **25MHz * (m** + **n**/**d**)

- **m** (the integer multipler) can range from **15** to **90**
- **n** (the numerator) can range from **0 to 1,048,575**
- **d** (the denominator) can range from **1** to **1,048,575**

Set up the clock divider

Once you have the PLLs set up, you can now divide that high frequency down to get the number you want for the output

Each output has its own divider. You can use the cleaner Integer-only divider:

clockgen.setupMultisynthInt(output, SI5351_PLL_x, SI5351_MULTISYNTH_DIV_x);

- For the **output** use 0, 1 or 2
- For the PLL input, use either **SI5351_PLL_A** or **SI5351_PLL_B**
- For the divider, you can divide by **SI5351_MULTISYNTH_DIV_4**, **SI5351_MULTISYNTH_DIV_6**, or **SI5351_MULTISYNTH_DIV_8**

Again, integer output will give you the cleanest clock. If you need more flexibility, use the fractional generator/divider:

clockgen.setupMultisynth(output, SI5351_PLL_x, div, n, d);

- For the **output** use 0, 1 or 2
- For the PLL input, use either **SI5351_PLL_A** or **SI5351_PLL_B**
- The final frequency is equal to the **PLL** / **(div + n/d)**
- **div** can range from **4** to **900**
- **n** can range from **0 to 1,048,575**
- **d** can range from **1 to 1,048,575**

Additional R Divider

If you need to divide even more, to get to the < 100 KHz frequencies, there's an additional R divider, that divides the output once more by a fixed number:

```
clockgen.setupRdiv(output, SI5351_R_DIV_x);
```

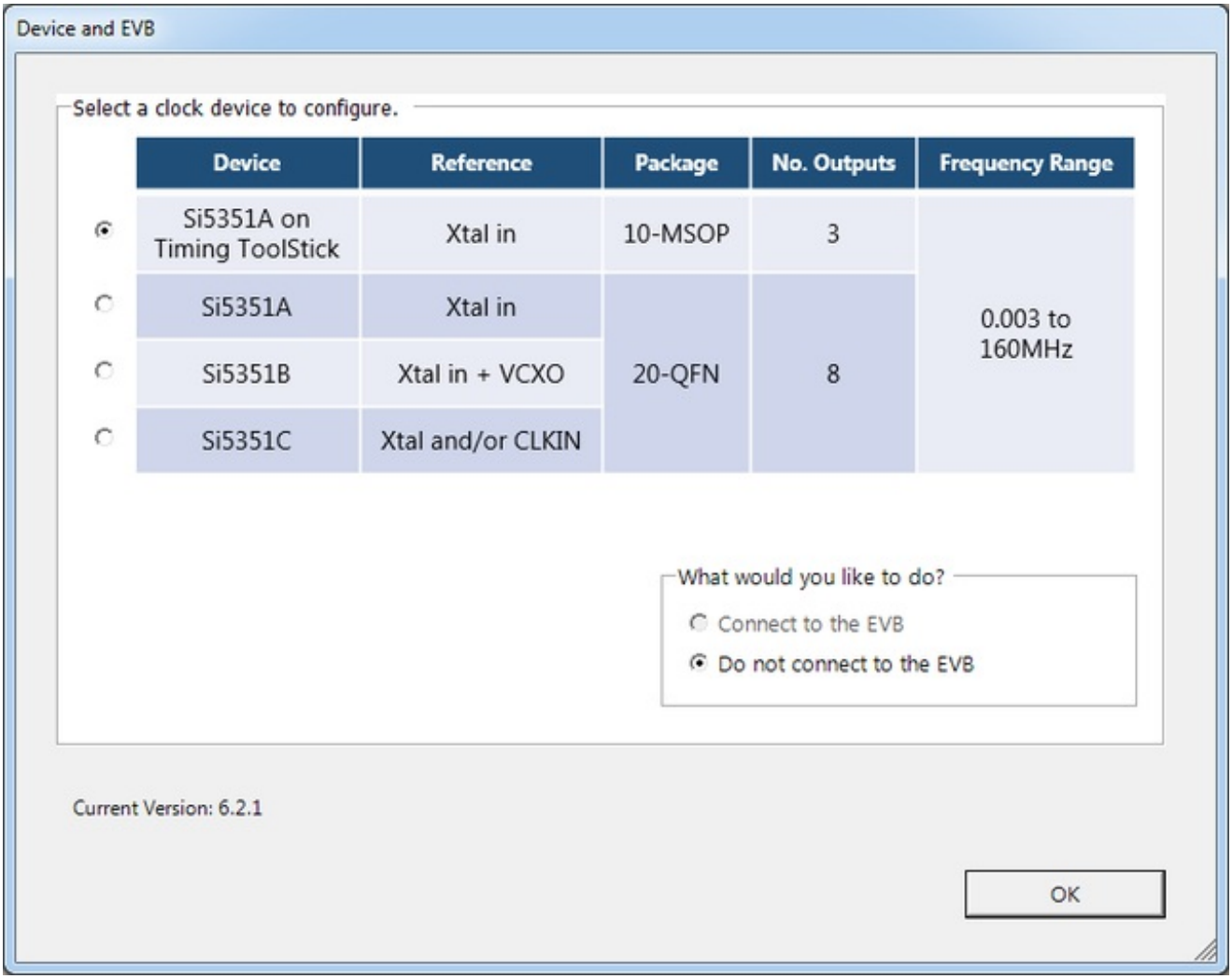
output is the clock output #
The R divider can be any of the following:

- SI5351_R_DIV_1
- SI5351_R_DIV_2
- SI5351_R_DIV_4
- SI5351_R_DIV_8
- SI5351_R_DIV_16
- SI5351_R_DIV_32
- SI5351_R_DIV_64
- SI5351_R_DIV_128

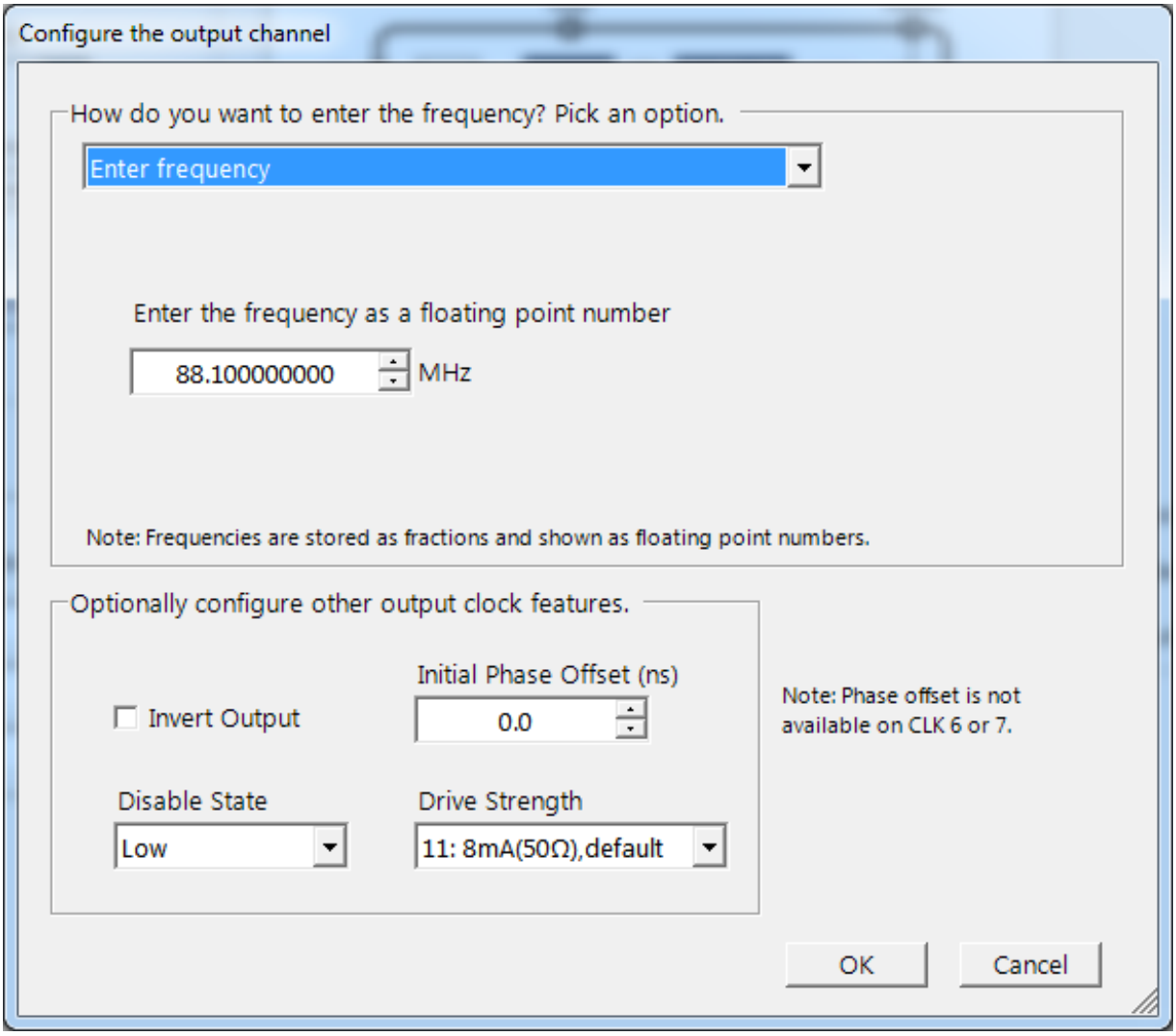
Software

As you can see, the annoying part here is figuring out the best choice for PLL multipler & divider! SiLabs has a desktop application called ClockBuilder (http://adafru.it/dPj) that can do some calculation of the PLL divider/multiplier for you. It's windows only, but you only need to use it once for calculation.

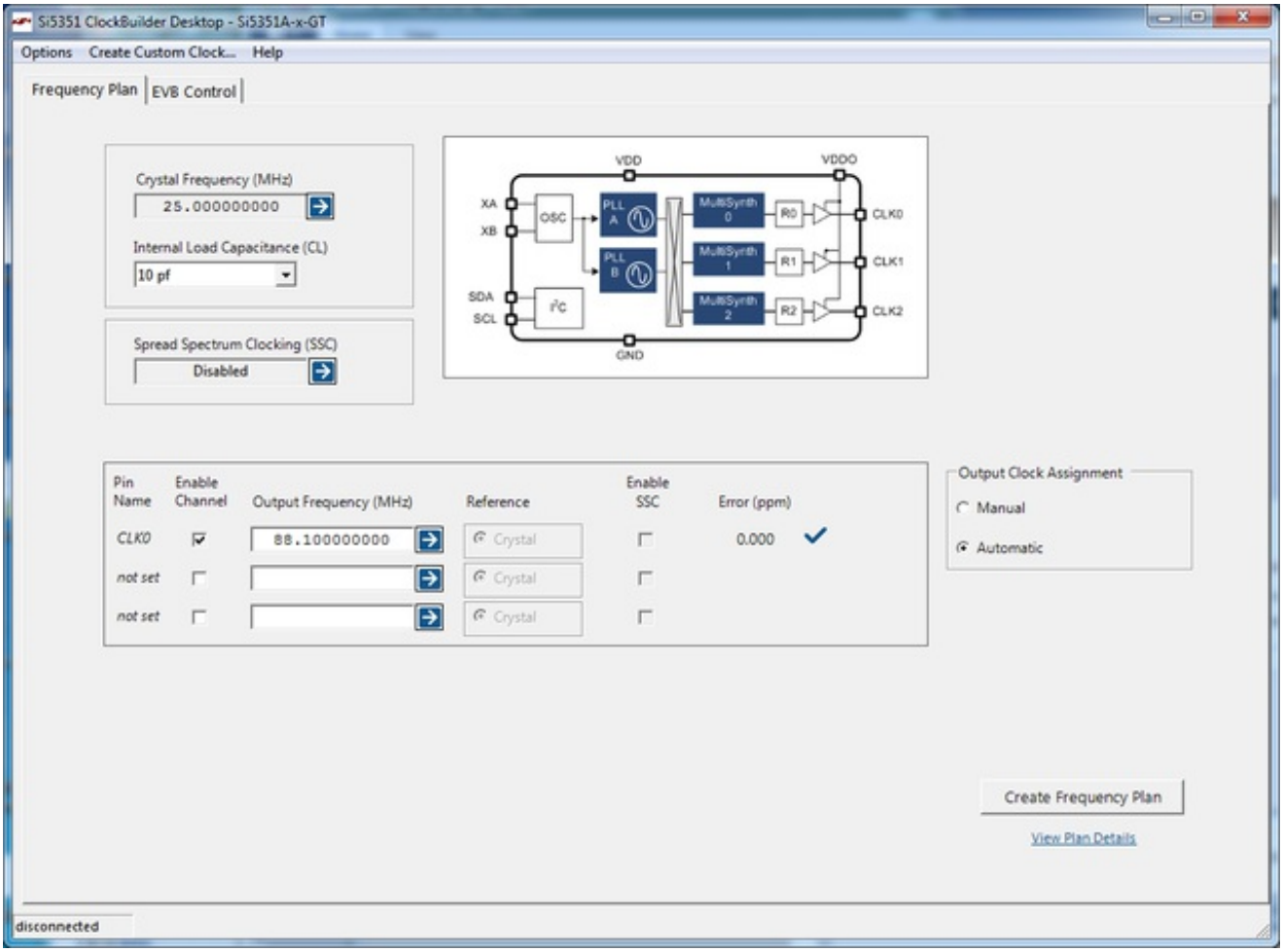
Install and run, select the **Si5351A** with 3 outputs, and **Do not connect to the EVB**



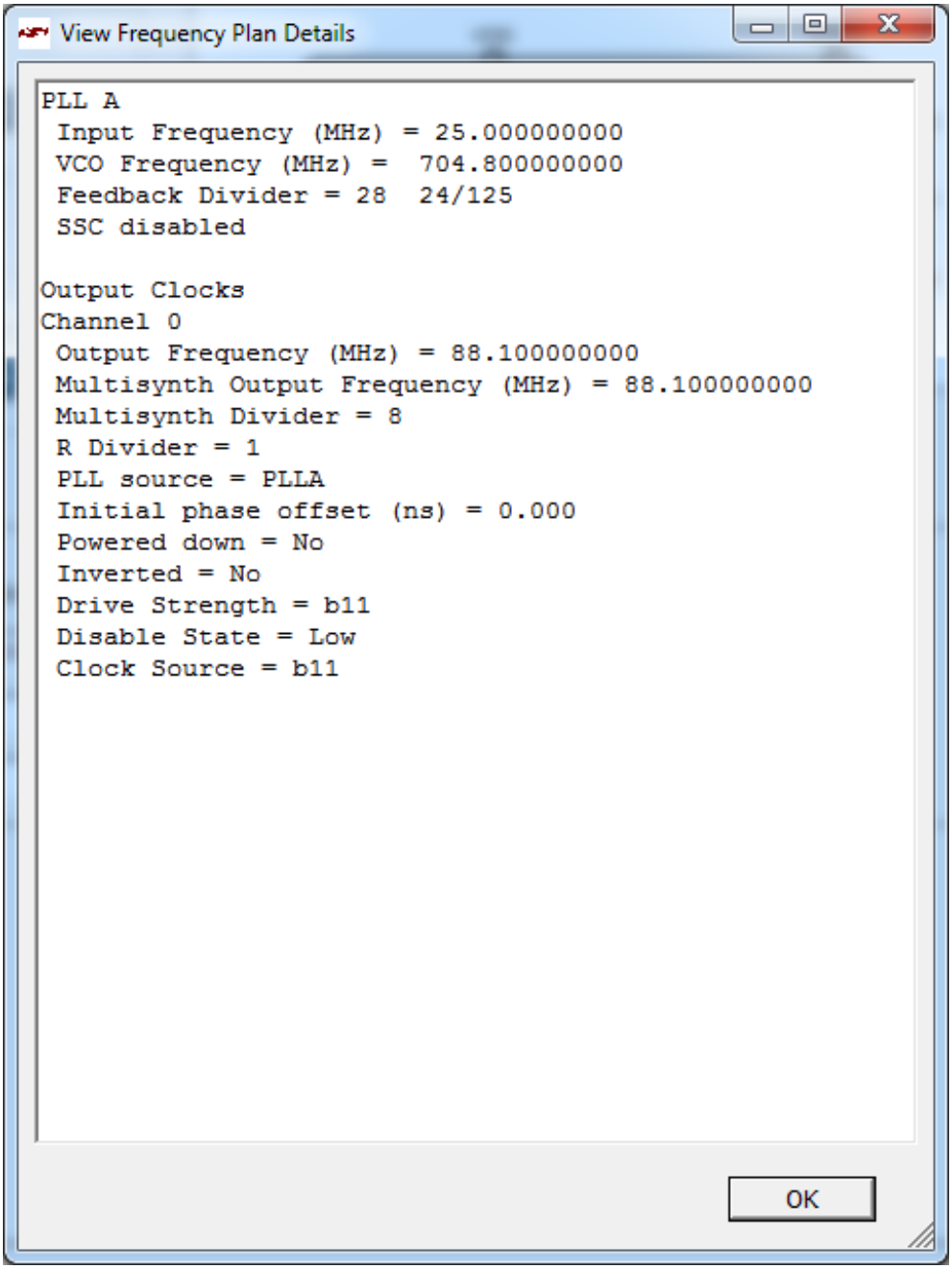
Enable the output you want, and set the frequency as floating point or fraction



Set up the crystal to be 25 MHz (the default is 27 MHz)



Click on **Create Frequency Plan** to see the PLL and divider setups!


View Frequency Plan Details
PLL A
Input Frequency (MHz) = 25.000000000
VCO Frequency (MHz) = 704.800000000
Feedback Divider = 28 24/125
SSC disabled
Output Clocks
Channel 0
Output Frequency (MHz) = 88.100000000
Multisynth Output Frequency (MHz) = 88.100000000
Multisynth Divider = 8
R Divider = 1
PLL source = PLLA
Initial phase offset (ns) = 0.000
Powered down = No
Inverted = No
Drive Strength = b11
Disable State = Low
Clock Source = b11
OK

Downloads

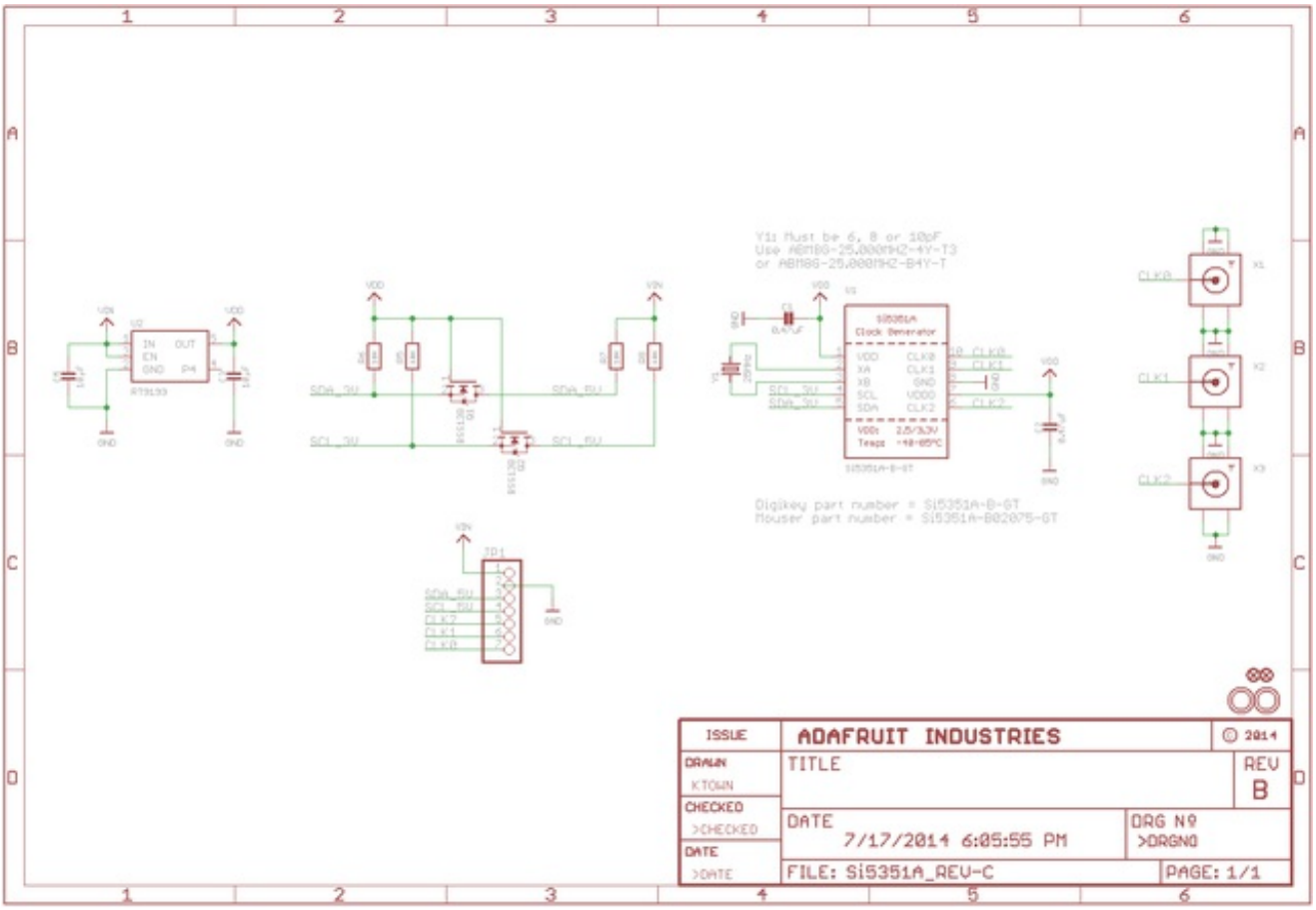
Software

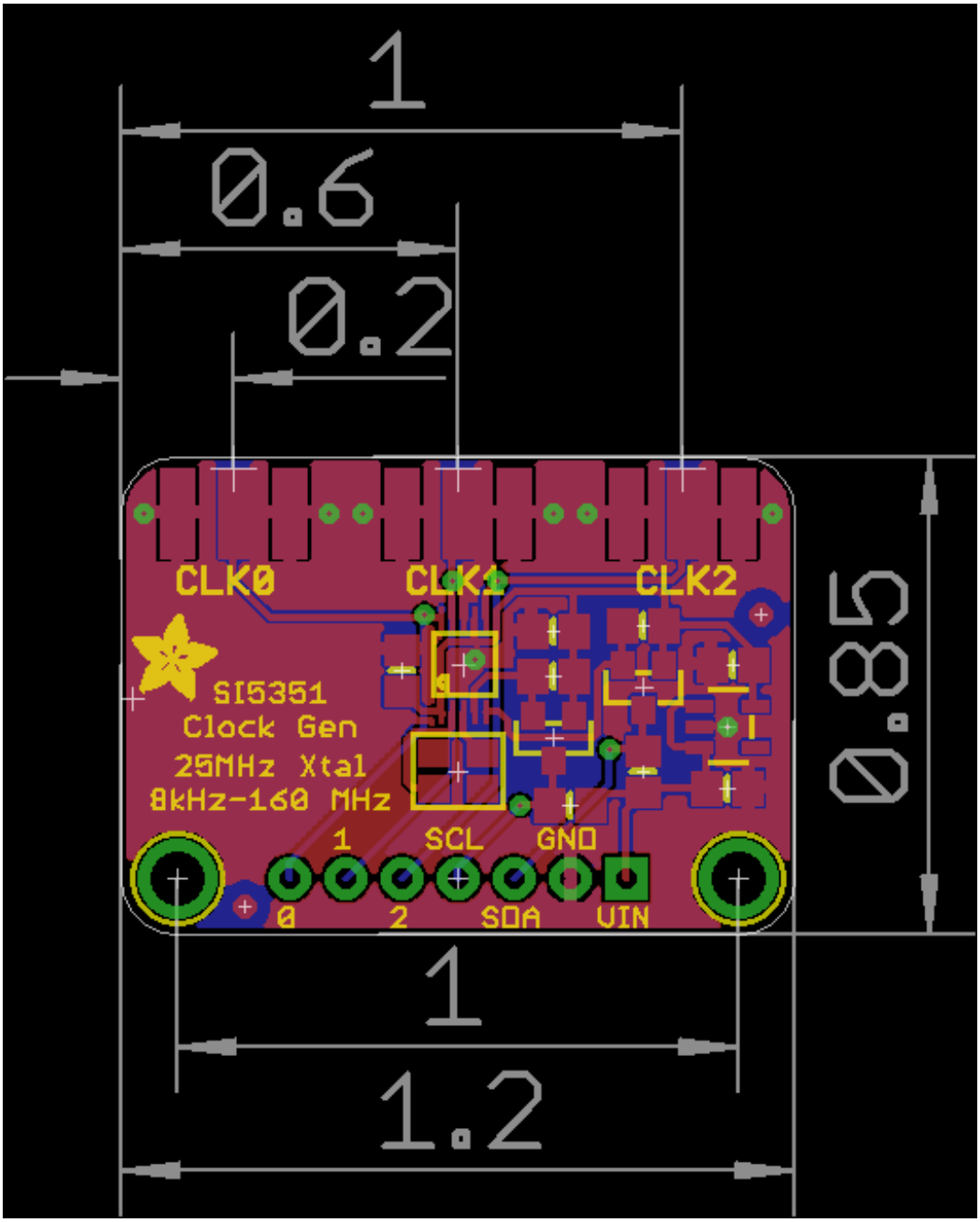
SiLabs has a desktop application called ClockBuilder (http://adafru.it/dPj) that can do some calculation of the PLL divider/multiplier for you.

Datasheet

- Si5351 Datasheet (http://adafru.it/dPk)
- Fritzing object in Adafruit Fritzing library (http://adafru.it/aP3)
- EagleCAD PCB files on GitHub (http://adafru.it/pBD)

Schematic and PCB Print




1
0.6
0.2
0.85
CLK0
CLK1
CLK2
SI5351
Clock Gen
25MHz Xtal
8kHz-160 MHz
1
SCL
GND
0
2
SDA
VIN
1
1.2