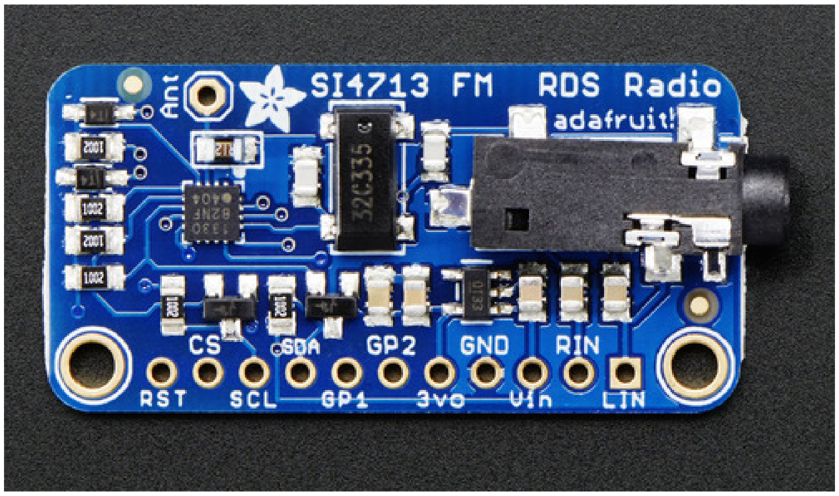
adafruit learning system

Adafruit Si4713 FM Radio Transmitter with RDS/RDBS Support

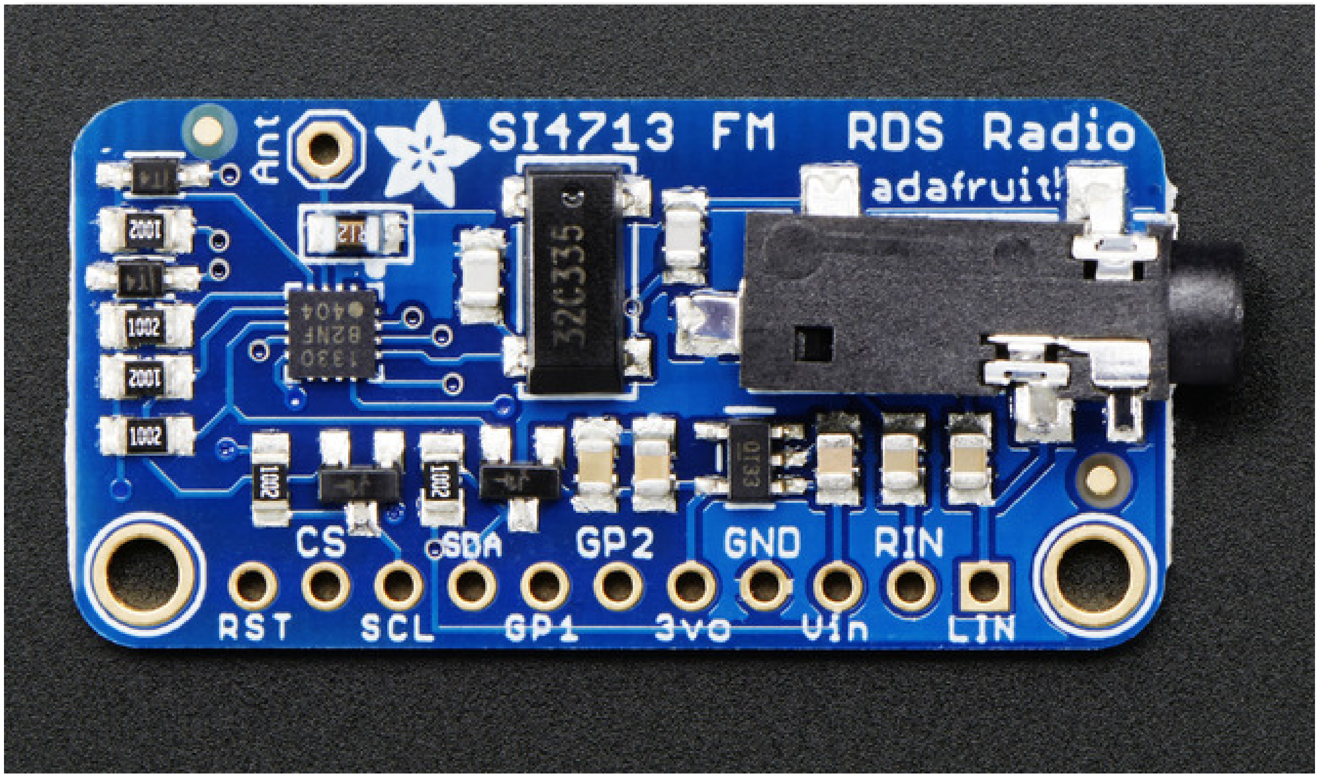
Created by lady ada



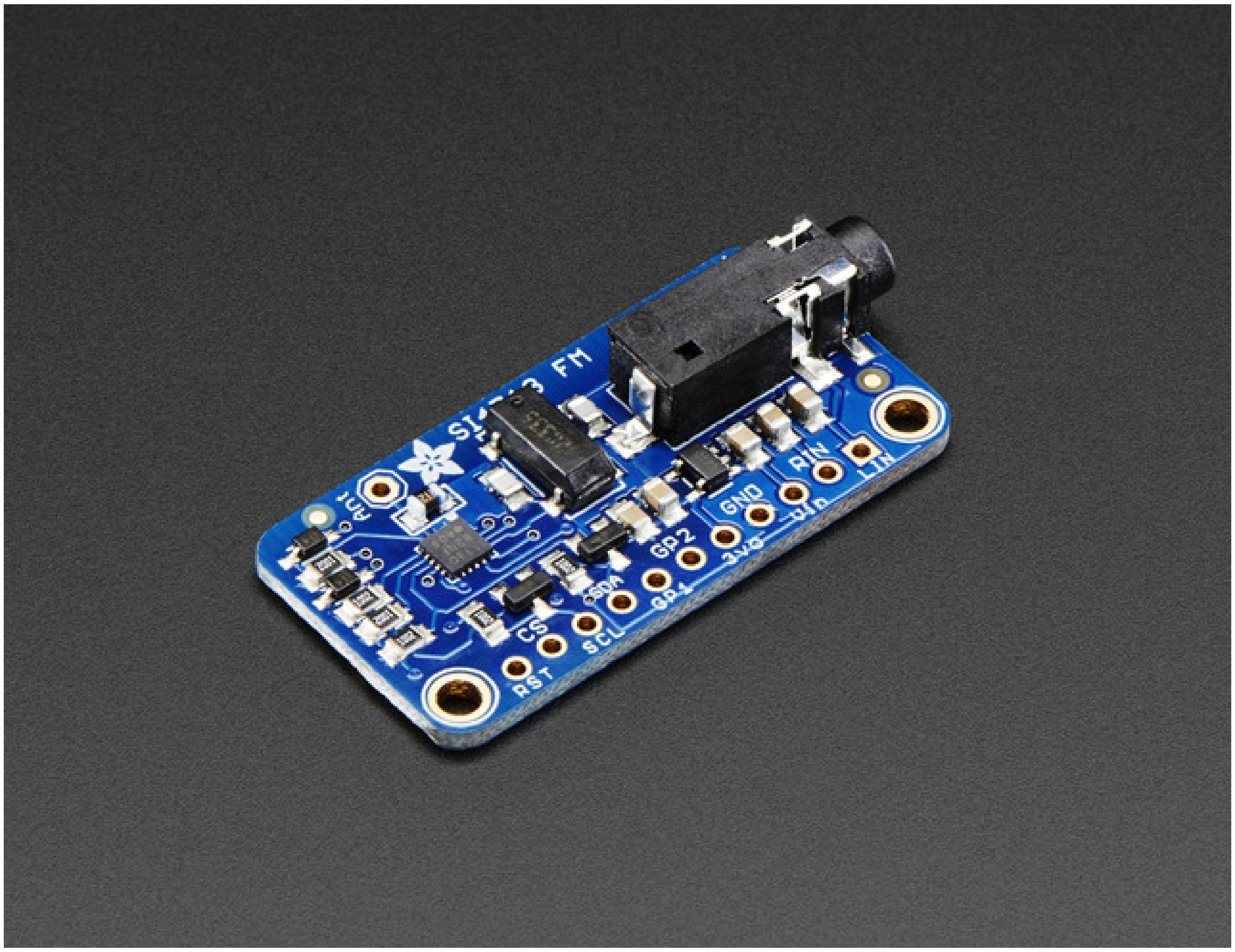
Last updated on 2014-07-09 11:00:12 AM EDT

Guide Contents

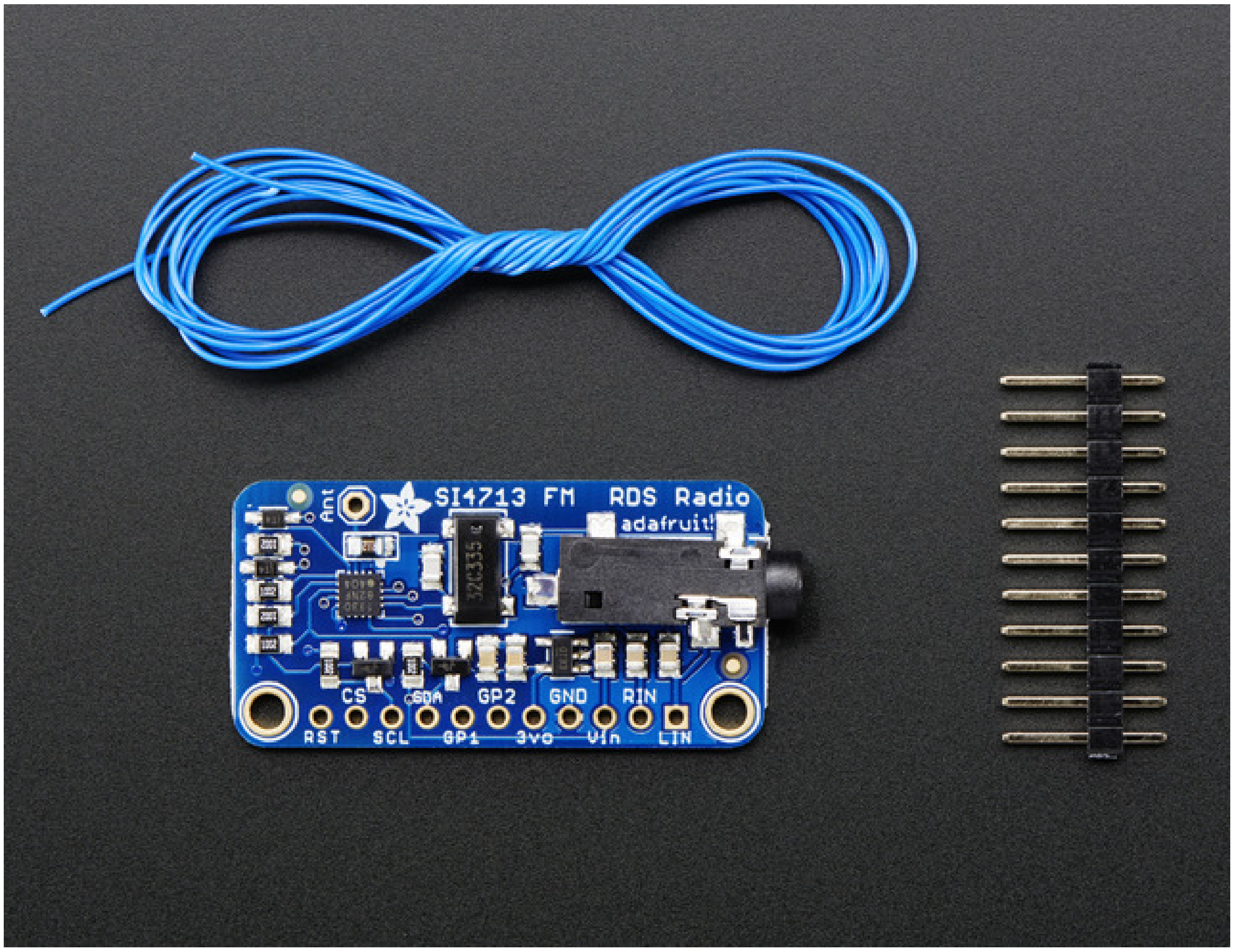
Overview



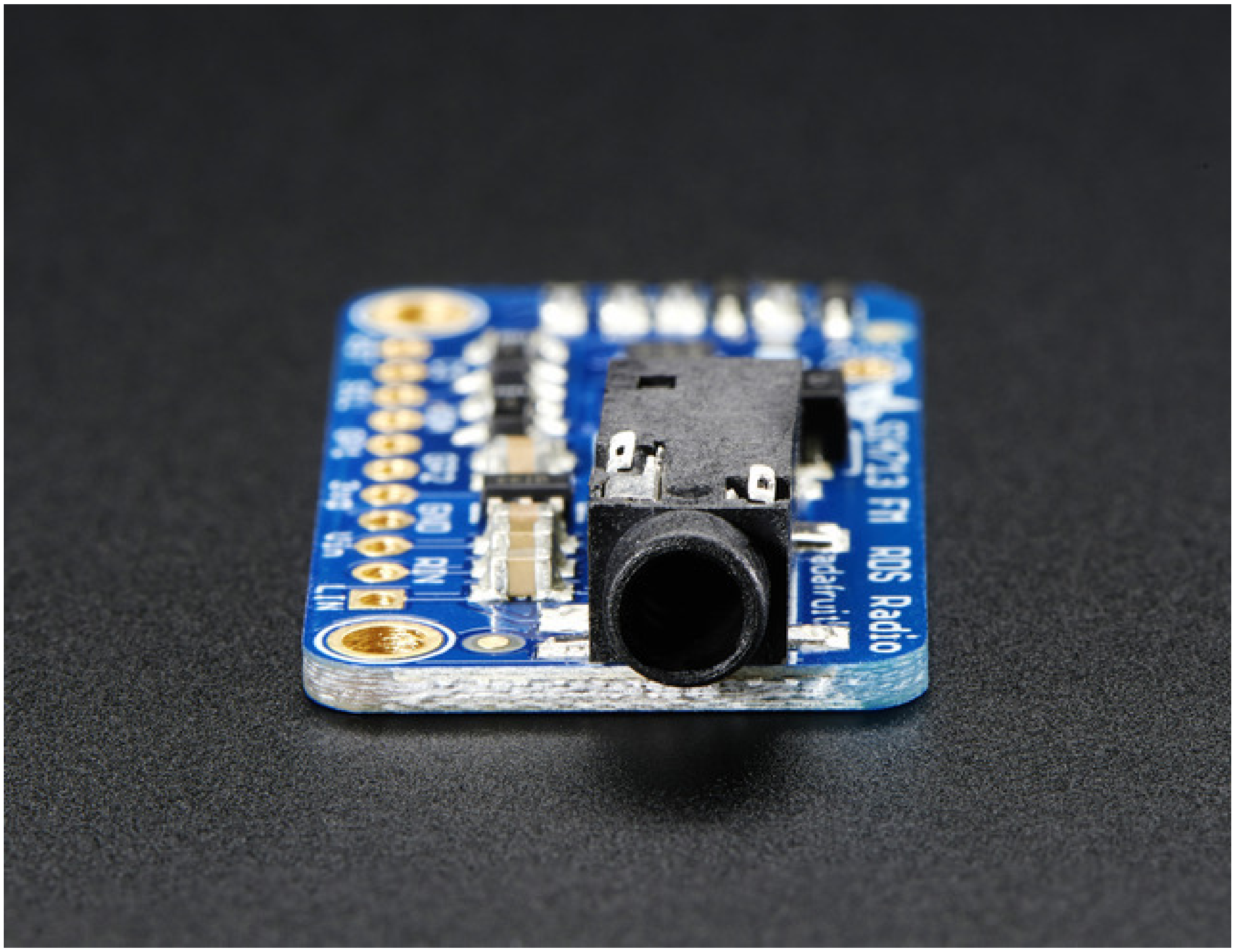
Yaaar! Become your very own pirate radio station with this FM radio transmitter. This breakout board, based on the best-of-class Si4713, is an all-in-one stereo audio FM transmitter that can also transmit RDS/RBDS data!



Wire up to your favorite microcontroller (we suggest an Arduino) to the I2C data lines to set the transmit frequency and play line-level audio into the stereo headphone jack. Boom! Now you **are** the media. Listen using any FM receiver such as your car or pocket radio receiver - this is an easy way to transmit audio up to about 10 meters / 30 feet away.



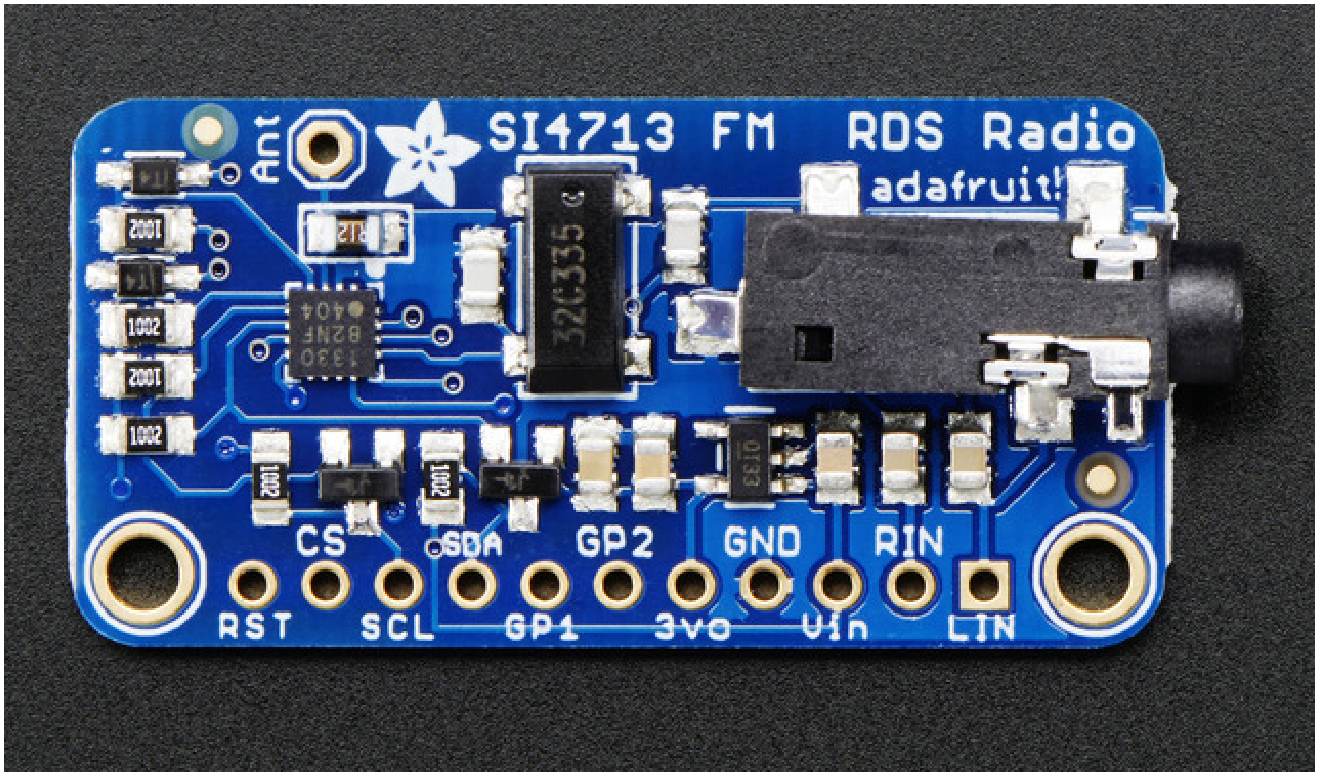
This transmitter even has RDS/RBDS support - that's text/data transmissions that many modern FM receivers support. (It's how some car radios can display the FM station and current song playing). You can transmit just about any text you want, set the station identifier as well as the 'freeform' buffer.



Best of all, you'll be up and running in minutes with our awesome Arduino library, example code and tutorial!

Pinouts

There's a couple pins on this here breakout, lets cover them all in groupings by 'type'



Audio Inputs

- **LIN** - this is the line level **LEFT** input. Its connected to the headphone jack as well but in case you want to wire directly without a chunky cable, pipe line level (~0.7 Vpp) audio into here. There's an AC blocking capacitor on board so it can be DC biased
- **RIN** - same **as LIN** but the **RIGHT** input.

Power Pins

- **Vin** - this is the power input pin. You can power the chip from 3-5VDC. Ideally you should use the same voltage you use for logic levels. For an Arduino, that's usually 5V
- **GND -** this is power and logic ground, connect to your microcontroller's ground pin
- **3Vo** - this is the output from the onboard regulator, 3.3V nominal. You can use this if you need up to 100mA of 3V regulated voltage

Interface Pins

The FM transmitter chip requires a microcontroller for setting it up unlike pure-analog solutions that have a tuning potentiometer. The trade off is some code is needed, but the output is digitally tuned so its much more precise.
Our codebase uses I2C to communicate. The chip supports SPI as well but it was annoying

enough to support just I2C so we don't have code examples for SPI!

All the interface input pins are 5V friendly, and can be used with 3-5V logic

- **RST** - This is the Reset pin. You must have this pin toggle before starting to communicate with the chip. When at logic 0, the chip is in reset.
- **CS** - This is the Chip select pin, used in SPI mode. It also determines the I2C address. When pulled high (it is by default) the I2C address is 0x63. If this pin is shorted to ground, the I2C address is 0x11
- **SCL** - this is the I2C clock pin, connect to SCL on your microcontroller.
- **SDA** - this is the I2C data pin, connect to SDA on your microcontroller.

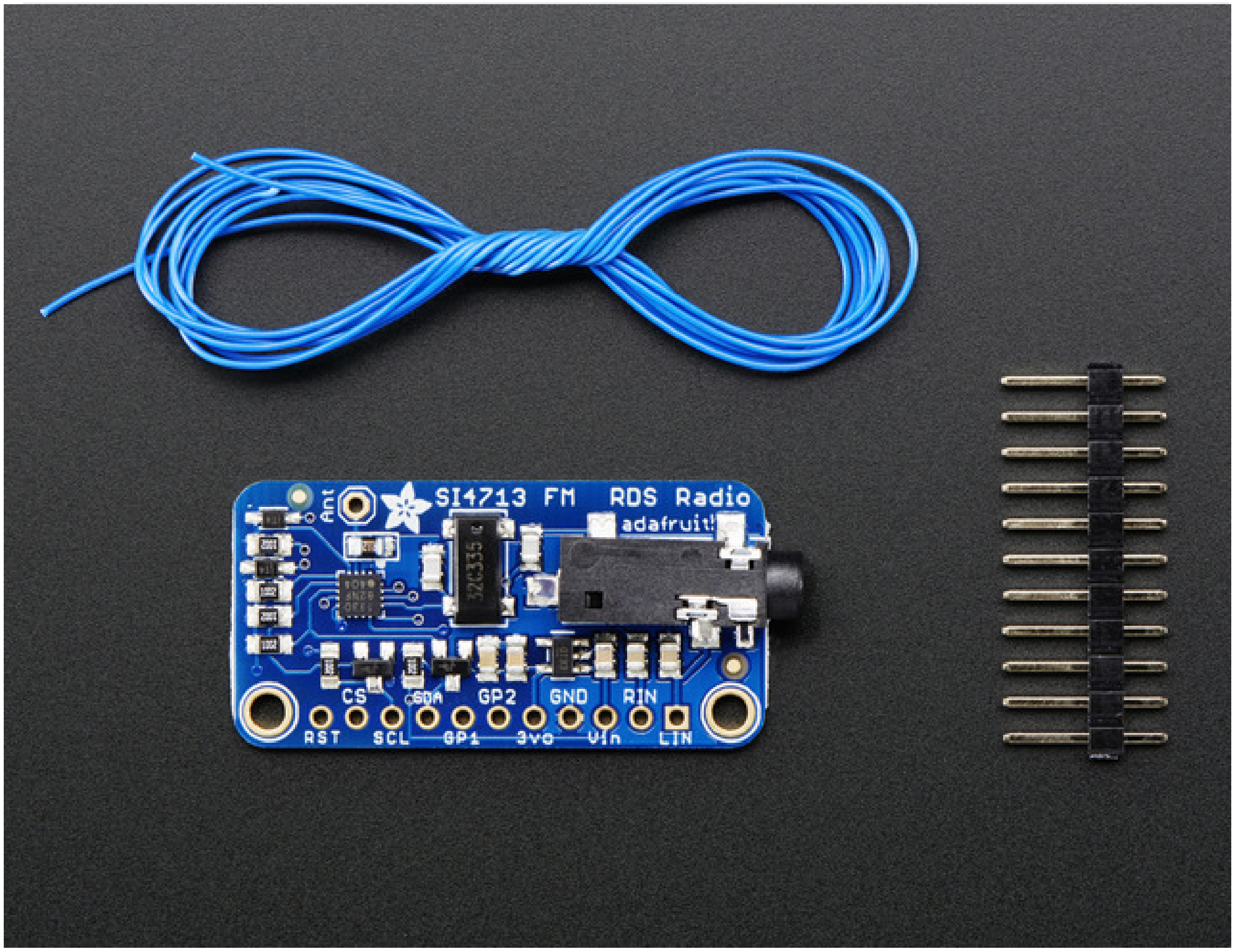
Extra GPIO Pins

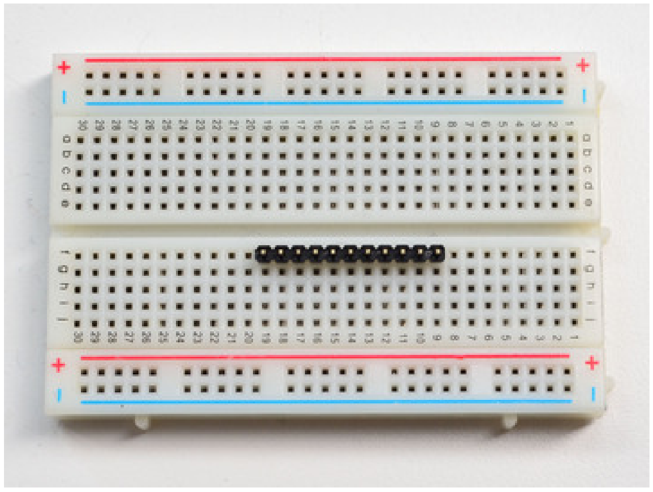
There's also two "GPIO" pins, you can use these to blink LEDs. The initial state of these pin sets up the chip for Analog Mode so don't short them to ground or VCC during reset. They are 3V output only!

- **GP1** - this is GPIO #1
- **GP2** - this is GPIO #2

GPIO #3 is used for the 32Khz clock generator onboard.

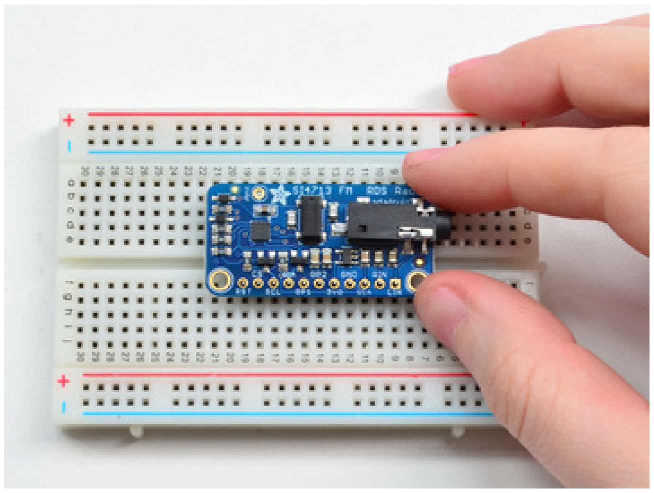
Assembly





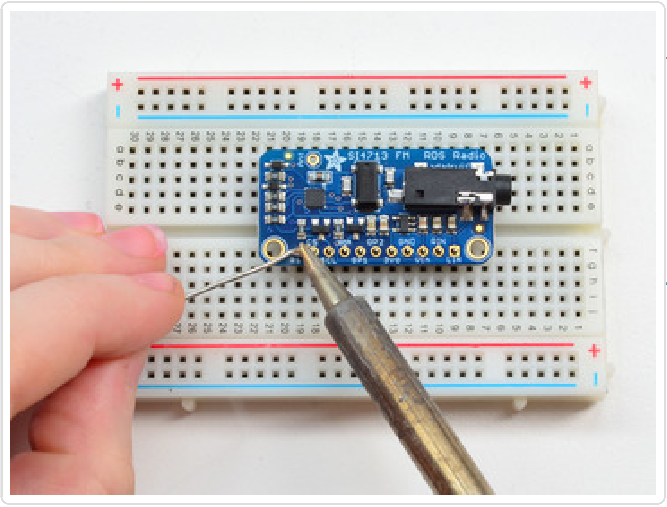
Prepare the header strip:

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:

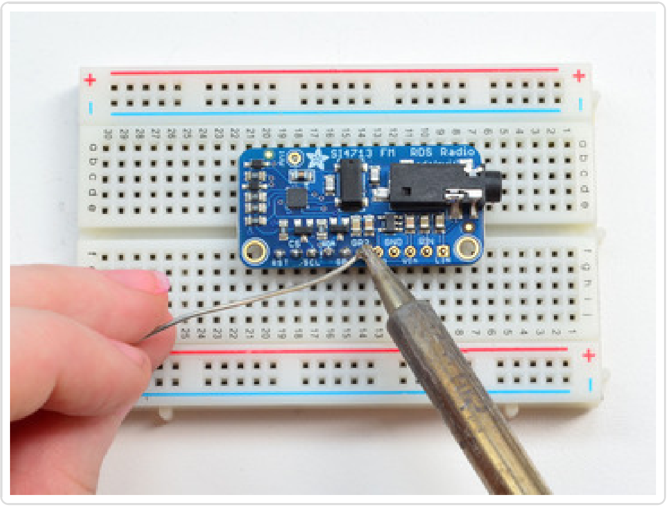
Place the breakout board over the pins so that the short pins poke through the breakout pads

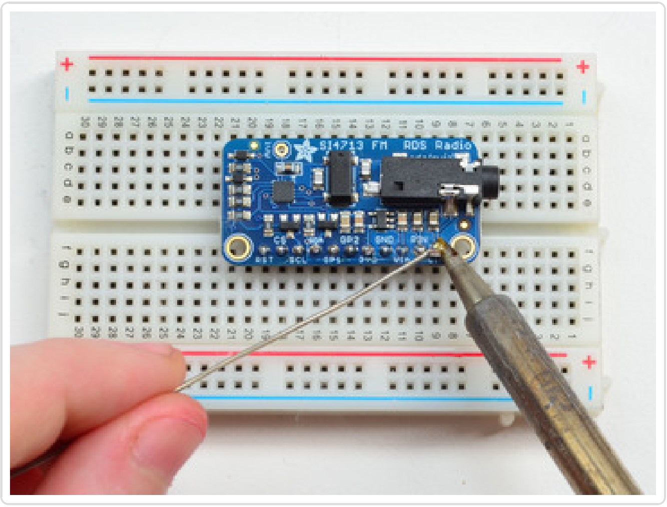


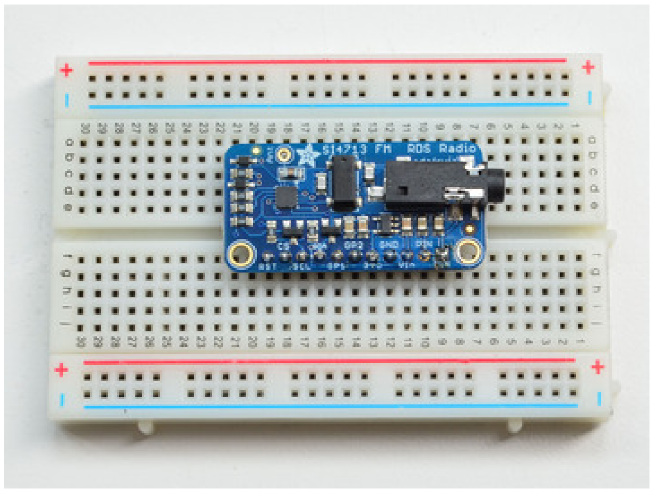
And Solder!

Be sure to solder all pins for reliable electrical contact.

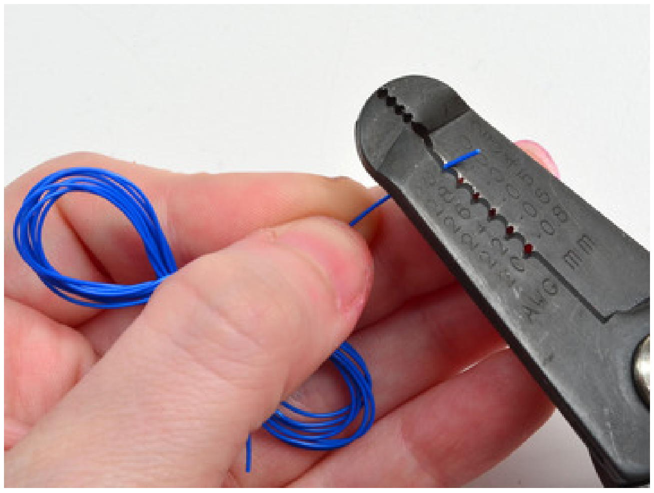
(For tips on soldering, be sure to check out our Guide to Excellent Soldering (http://adafru.it/aTk)).



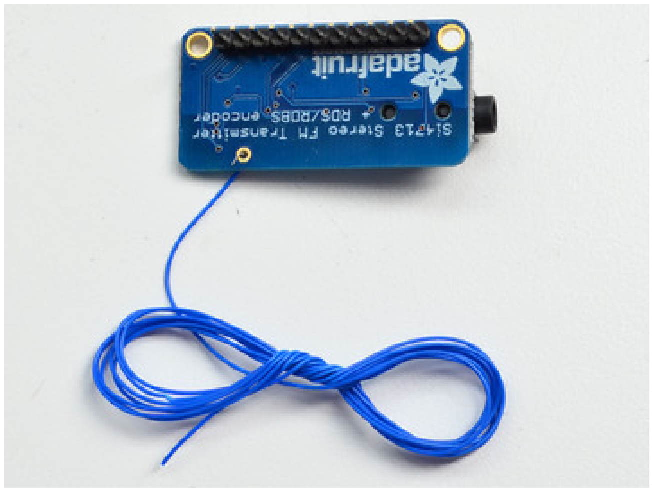




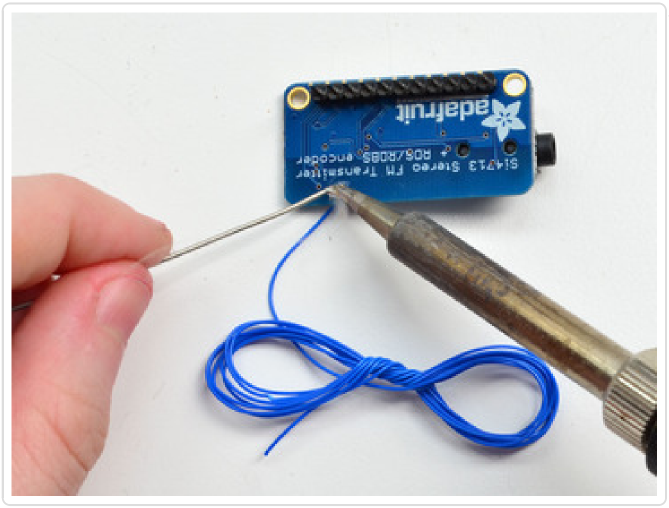
You're done! Check your solder joints visually and continue onto the antenna



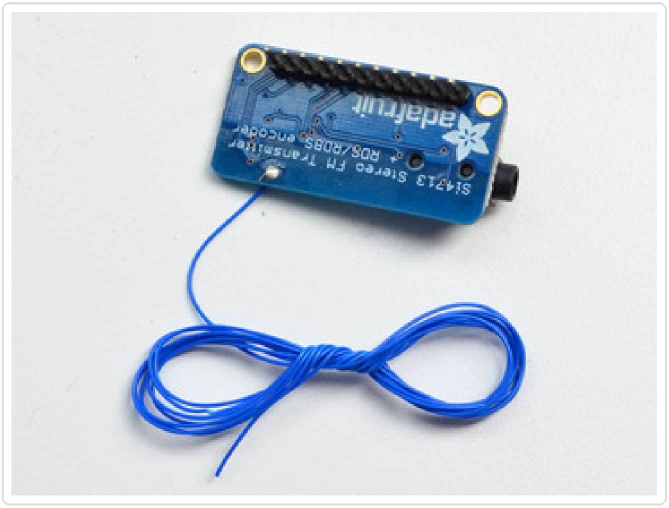
An antenna is required! We provide a 1meter long wire but you can also use a shorter or longer piece as desired.
Strip a few mm from the end



Hook the exposed wire end into the **ANT** hole



Solder it in!



Done!

Test & Usage

Arduino Wiring

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, then port the code - once the low level i2c functions are adapted the rest should 'fall into place'

(http://adafru.it/dBn)

(http://adafru.it/dBo)

- 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**
- Connect the **RST** pin to digital **12** - you can change this later but we want to match the tutorial for now

The Si4713 has a default I2C address of **0x63** - you can change it to 0x11 by connecting **CS** to ground but don't do that yet! Get the demo working first before making changes

Download Adafruit_Si4713

To begin reading sensor data, you will need to download Adafruit_Si4713 Library from our github repository (http://adafru.it/dBp). You can do that by visiting the github repo and manually downloading or, easier, just click this button to download the zip

Download Adafruit_Si4713
Arduino Library

http://adafru.it/dBq

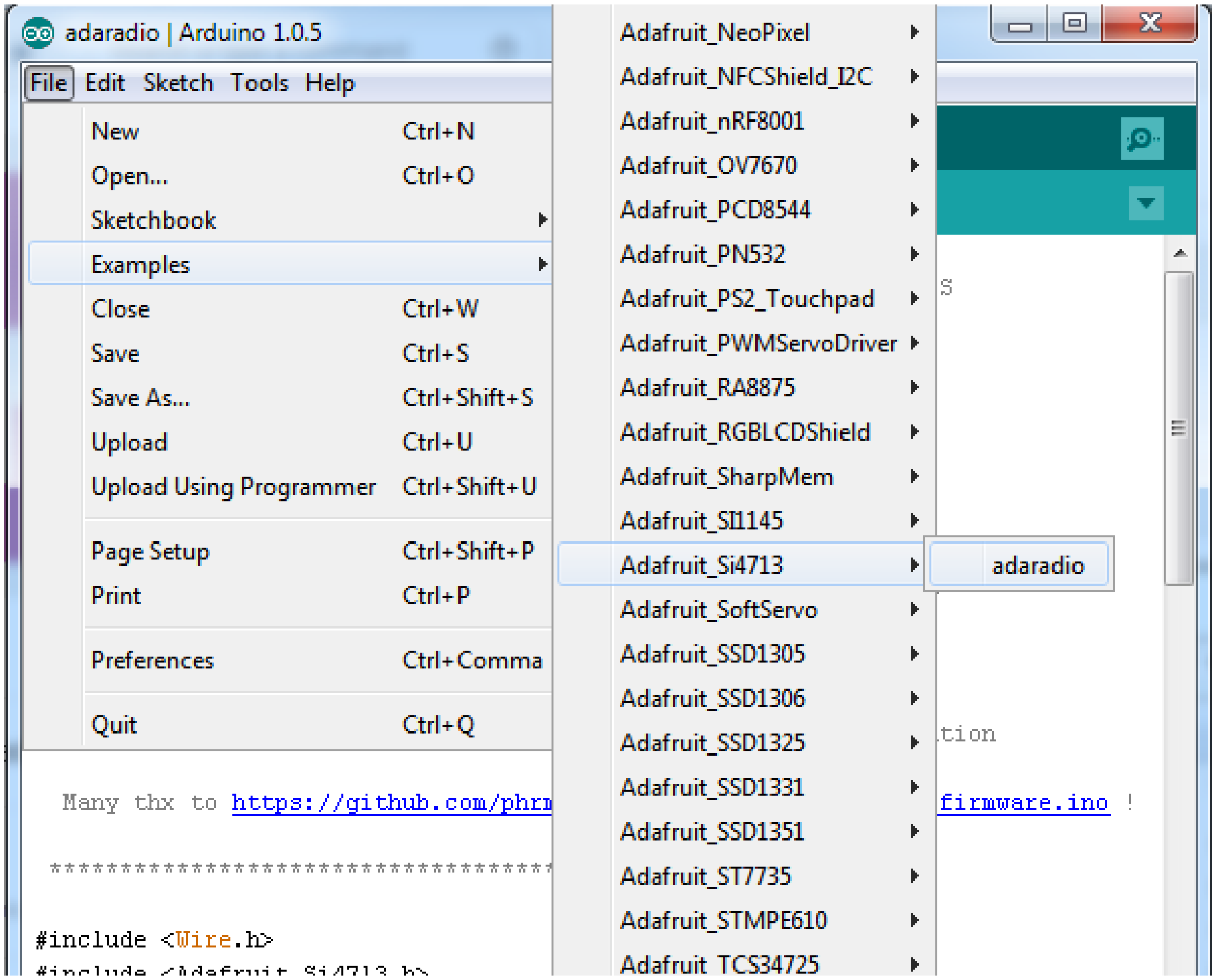
Rename the uncompressed folder **Adafruit_Si4713** and check that the **Adafruit_Si4713** folder contains **Adafruit_Si4713.cpp** and **Adafruit_Si4713.h**

Place the **Adafruit_Si4713** 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

Open up **File->Examples->Adafruit_Si4713->HTU21DFtest** and upload to your Arduino wired up to the sensor



```
 Many thx to https://github.com/phrm
 ***********************************

#include <Wire.h>
```

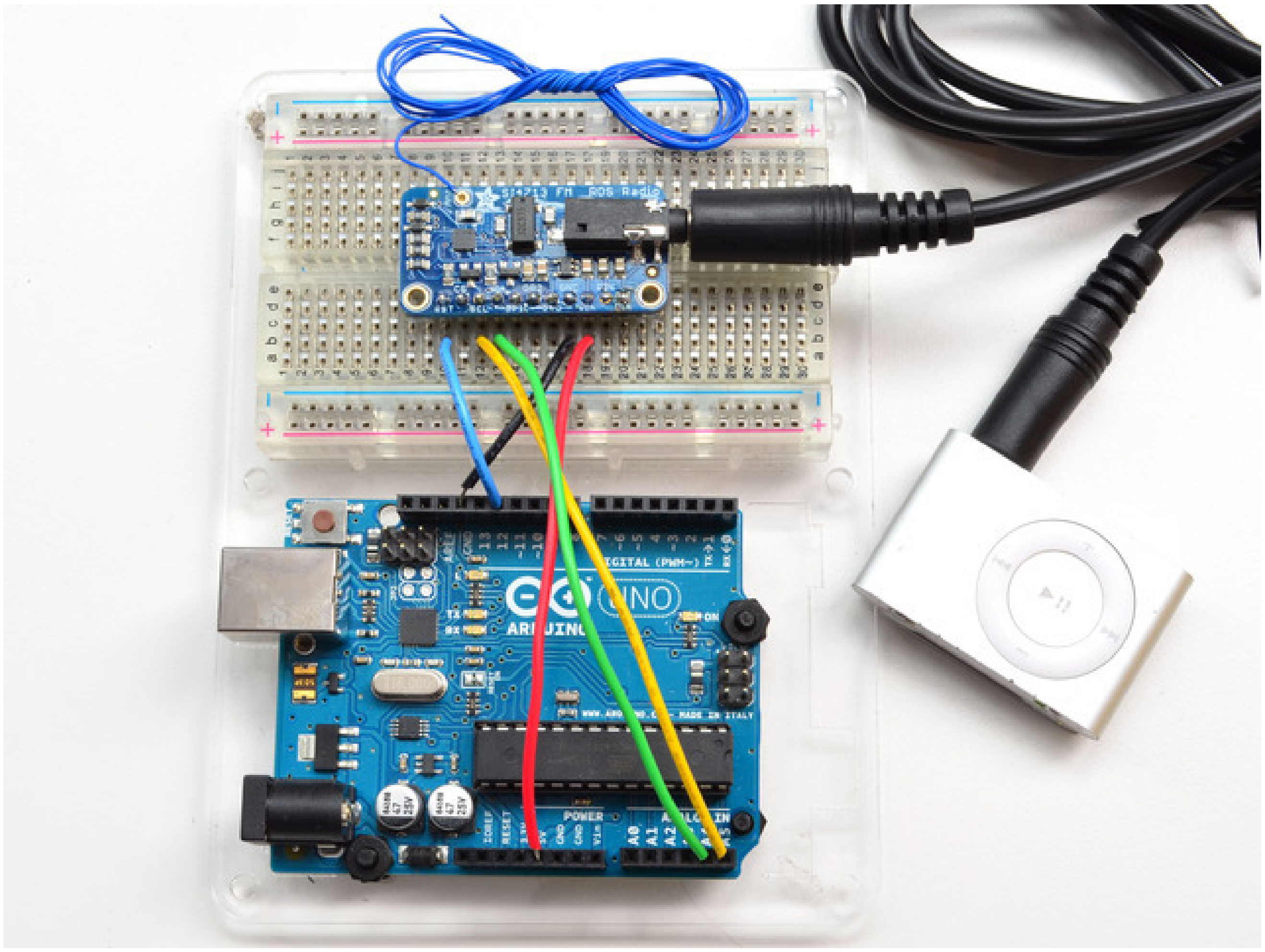
You may want to update the FM station transmission. By default the library transmits on 102.3MHz FM, but that might be 'taken' in your area.

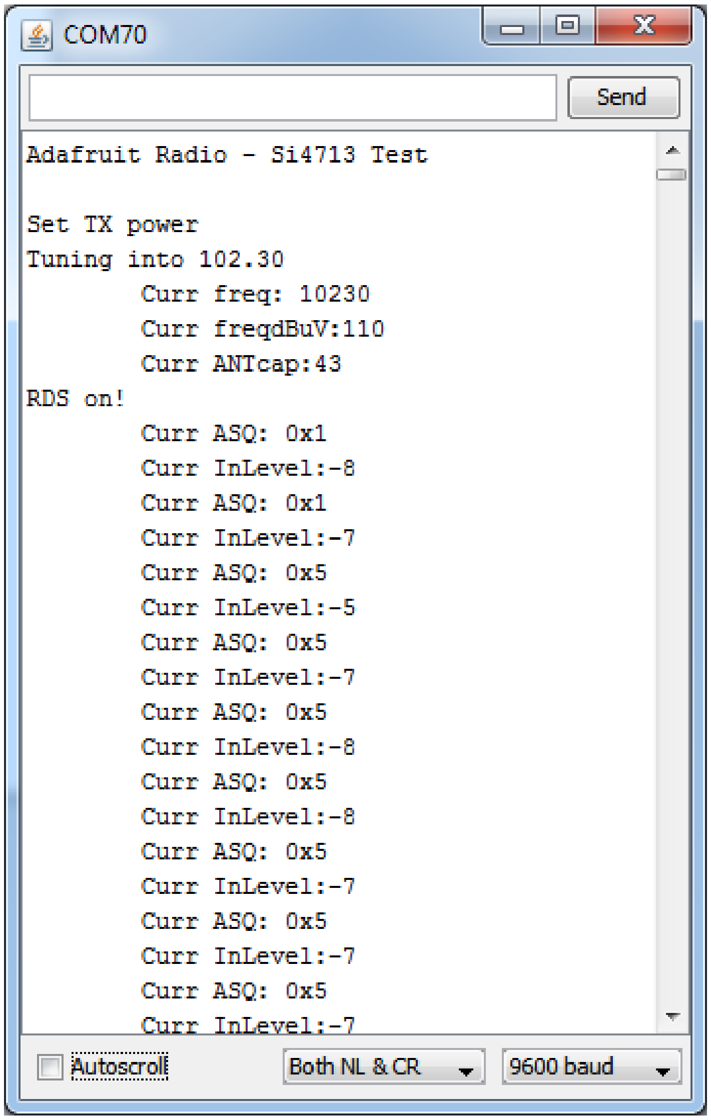
Find this line

> **#define FMSTATION 10230 // 10230 == 102.30 MHz**

And change it to an unused frequency. This number is in 10KHz so for example 88.1MHz is written as 8810

Upload it to your Arduino and open up the Serial console at 9600 baud



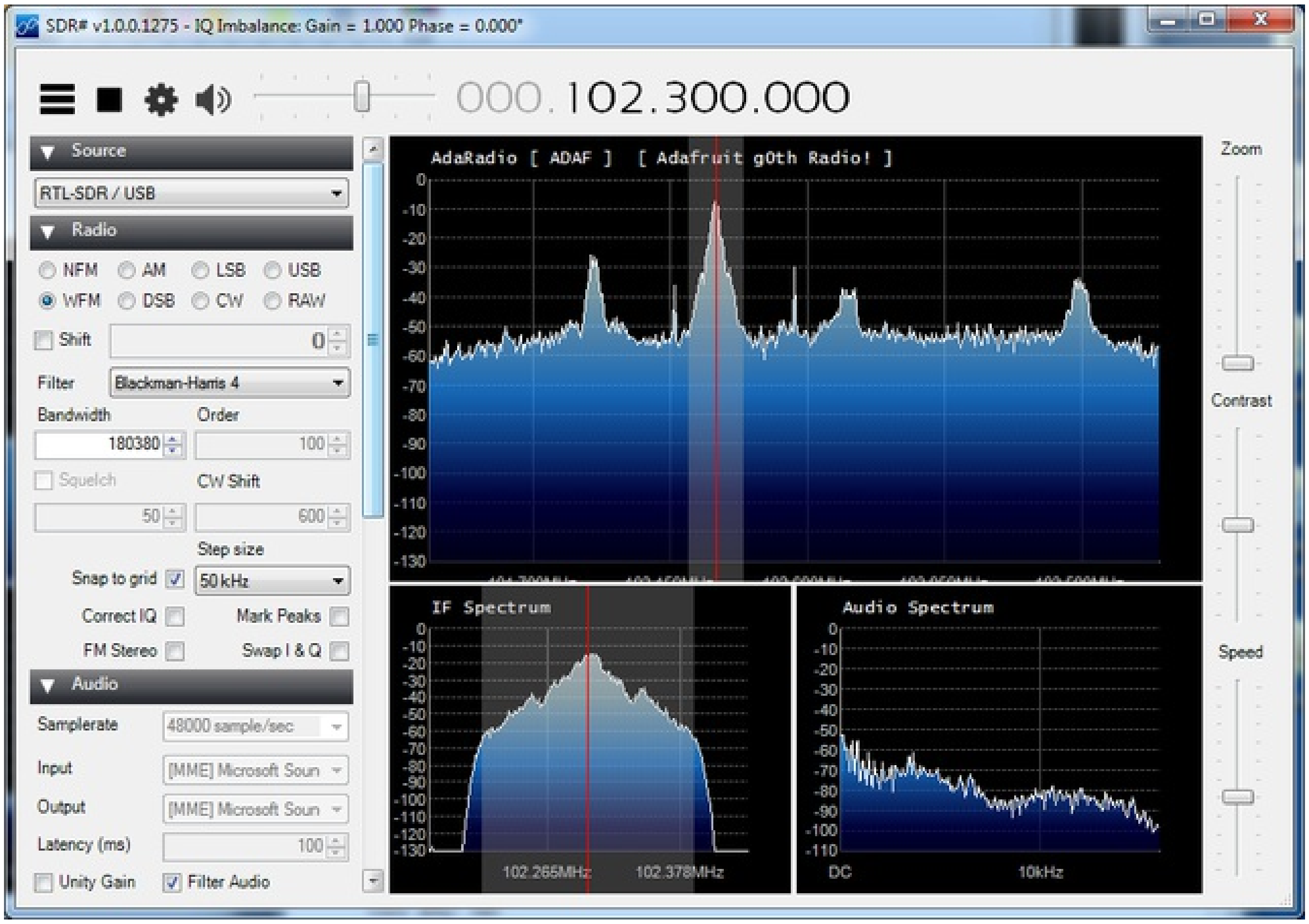


```
Adafruit Radio - Si4713 Test

Set TX power
Tuning into 102.30
        Curr freq: 10230
        Curr freqdBuV:110
        Curr ANTcap:43
RDS on!
        Curr ASQ: 0x1
        Curr InLevel:-8
        Curr ASQ: 0x1
        Curr InLevel:-7
        Curr ASQ: 0x5
        Curr InLevel:-5
        Curr ASQ: 0x5
        Curr InLevel:-7
        Curr ASQ: 0x5
        Curr InLevel:-8
        Curr ASQ: 0x5
        Curr InLevel:-8
        Curr ASQ: 0x5
        Curr InLevel:-7
        Curr ASQ: 0x5
        Curr InLevel:-7
        Curr ASQ: 0x5
        Curr InLevel:-7
```

As long as you get to the **RDS On!** message that means everything works, pipe some audio into the 3.5mm jack and make sure you see the **InLevel** audio volume range from 0 to about -10 (dB)

The fastest way to test the RDS message sending is using an RTL-SDR (that's how we debugged the breakout!) (http://adafru.it/dBr) or a phone/radio that can do RDS decoding


SDR# v1.0.0.1275 - IQ Imbalance: Gain = 1.000 Phase = 0.000°

000.102.300.000

Source: RTL-SDR / USB

Radio: ○ NFM ○ AM ○ LSB ○ USB [✓ WFM] ○ DSB ○ CW ○ RAW

☐ Shift 0

Filter: Blackman-Harris 4

Bandwidth: 180380 Order: 100

☐ Squelch 50 CW Shift: 600

Step size: 50 kHz

Snap to grid ☑

Correct IQ ☐ Mark Peaks ☐

FM Stereo ☐ Swap I & Q ☐

Audio

Samplerate: 48000 sample/sec

Input: [MME] Microsoft Soun

Output: [MME] Microsoft Soun

Latency (ms): 100

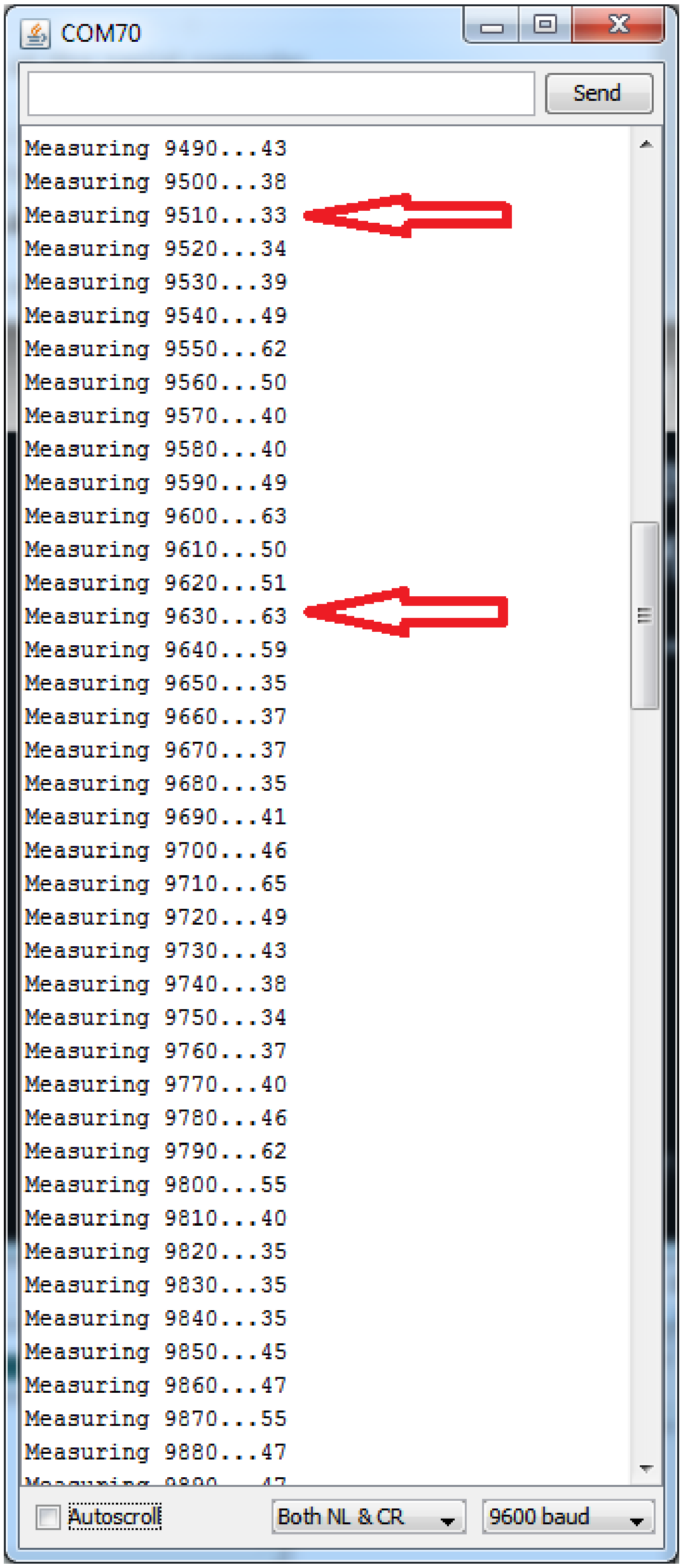
☐ Unity Gain ☑ Filter Audio

Using the RPS Scanning function

The Si4713 has the ability 'scan' the FM band and measure the input power. You can use the RPS functionality to locate a good unused station. Find this section in the adaradio demo and uncomment the for loop:

```
  // Uncomment below to scan power of entire range from 87.5 to 108.0 MHz
/*
  for (uint16_t f  = 8750; f<10800; f+=10) {
   radio.readTuneMeasure(f);
   Serial.print("Measuring "); Serial.print(f); Serial.print("...");
   radio.readTuneStatus();
   Serial.println(radio.currNoiseLevel);
   }
*/
```

Reupload and look at the serial console:



```
Measuring 9490...43
Measuring 9500...38
Measuring 9510...33
Measuring 9520...34
Measuring 9530...39
Measuring 9540...49
Measuring 9550...62
Measuring 9560...50
Measuring 9570...40
Measuring 9580...40
Measuring 9590...49
Measuring 9600...63
Measuring 9610...50
Measuring 9620...51
Measuring 9630...63
Measuring 9640...59
Measuring 9650...35
Measuring 9660...37
Measuring 9670...37
Measuring 9680...35
Measuring 9690...41
Measuring 9700...46
Measuring 9710...65
Measuring 9720...49
Measuring 9730...43
Measuring 9740...38
Measuring 9750...34
Measuring 9760...37
Measuring 9770...40
Measuring 9780...46
Measuring 9790...62
Measuring 9800...55
Measuring 9810...40
Measuring 9820...35
Measuring 9830...35
Measuring 9840...35
Measuring 9850...45
Measuring 9860...47
Measuring 9870...55
Measuring 9880...47
```

The larger the number the higher the transmission power. For example, 96.3MHz is a higher number than the others (FYI, its Univision 96.3 FM (http://adafru.it/dBs)!) whereas 95.1 MHz is nice as low, that's not used for any transmission. Try to find a number that's also not surrounded by high numbers, since it can get 'drowned out' by the nearby frequencies.

Library Reference

Radio Transmitter control

Start out by initializing the Si4713 chipset with

> **begin()**

This will return true if the radio initialized, and false if the radio was not found. Check your wiring if its not 'showing up'

Then you can turn on the radio transmitter with

> setTXpower(txpwr)

the txpwr number is the dBµV transmission power. You can set this to 88-115dBµV or 0 (for off)

Of course, you'll want to tune the transmitter! Do that with

> **tuneFM(*freq*)**

That will set the output frequency, in 10's of KHz. So if you want to tune to 101.9 the frequency value is 10190

You can check in on the radio with

> **readTuneStatus()**

Whcih will set the *currFreq* *currdBuV* adnd *currAntCap* variables in the radio object. The first two are the frequency and power output, the third variable is the tuning antenna capacitor it set for the best output. This number will vary with antenna size and frequency.

RPS (Radio Power Sensing)

This function is used with two procedures.

> **readTuneMeasure(freq)**

begins the measurement, freq is in units of 10KHz so 88.1MHz is written in as 8810
Then you have to call

> **readTuneStatus()**

which will wait until the chip has measured the data and stick it into the *currNoiseLevel* variable

RDS/RBDS (Radio Data Broadcast)

The Si4713 has great support for sending RDS data and we made it real easy too. Initialize the subsystem with

> **beginRDS()**

Then you can set the "station name" with

> **setRDSstation("AdaRadio")**

The radio station name is up to 8 characters
You can also send the main buffer which usually contains the song name/artist.

> **setRDSbuffer("Adafruit g0th Radio!")**

You can send up to 32 characters, but you can continuously send new data, just wait a few seconds before each data rewrite so the listener's radio has received all the data

GPIO Control

There's two GPIO pins you can use to blink LEDs. They are **GPIO1** and **GPIO2** - GPIO3 is used for the oscillator. To set them to be outputs call

> **setGPIOctrl(*bitmask*)**

where the bitmask has a 1 bit for each of the two pins. For example to set GPIO2 to be an output use setGPIOctrl((1<<2)) to set both outputs, use setGPIOctrl((1<<2) || (1<<1))

Then you can set the output with

> **setGPIO(*bitmask*)**

same idea with the bitmask, to turn both on, use setGPIOctrl((1<<2) || (1<<1)). To turn GPIO2 on and GPIO1 off, setGPIOctrl(1<<2)

Advanced!

We, by default, use the built-in AGC (auto-gain control) system so the audio level is maxed out. This may be annoying to you if have a good quality line level and the volume is fluctuating (it should be quiet, but isnt)

in the **Adafruit_Si4713.cpp** file find these lines

> **//setProperty(SI4713_PROP_TX_ACOMP_ENABLE, 0x02); // turn on limiter, but no dynamic ranging**
> **setProperty(SI4713_PROP_TX_ACOMP_ENABLE, 0x0); // turn on limiter and AGC**

and uncomment the first one, and comment the second. This will turn off the AGC

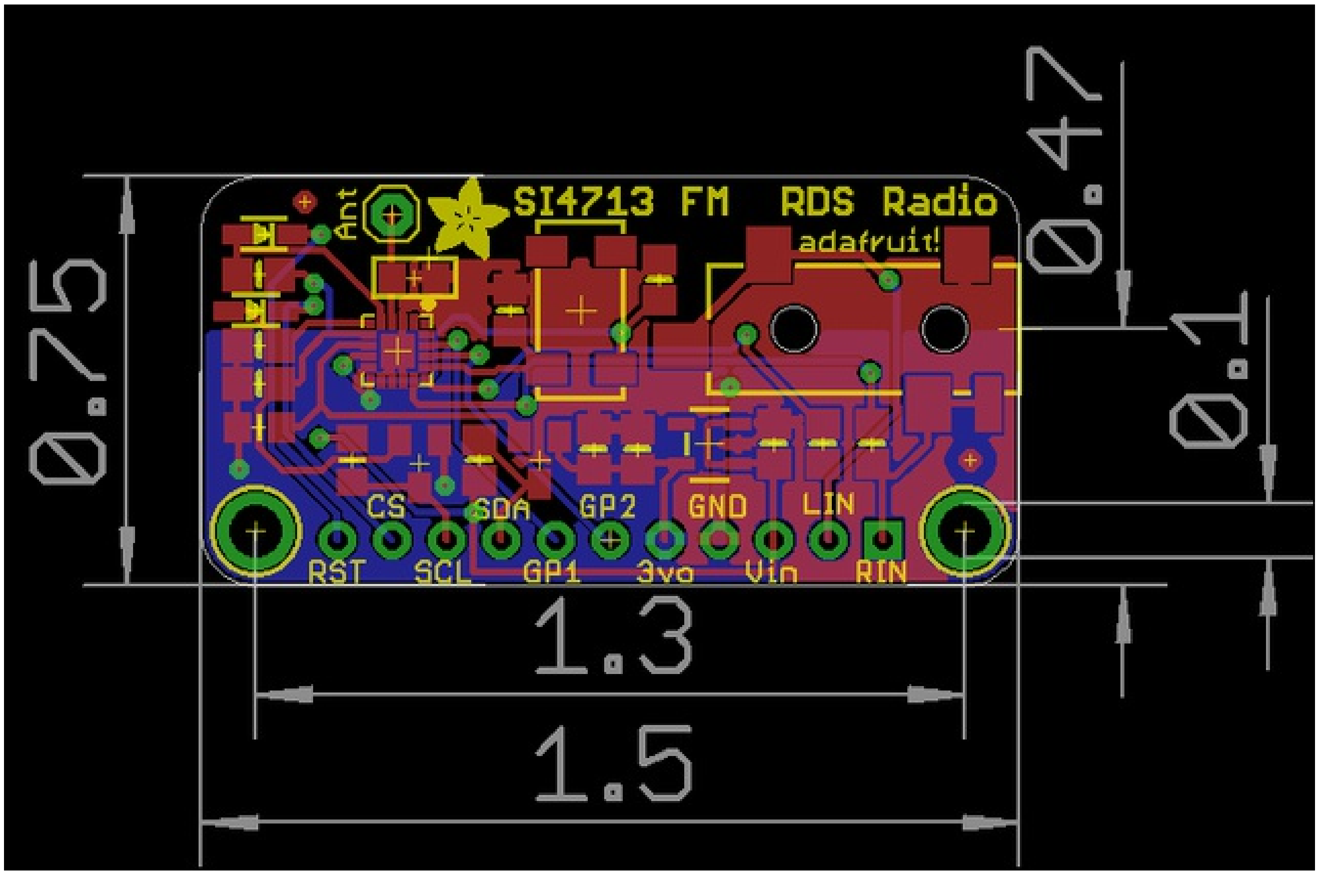
Downloads

Datasheets

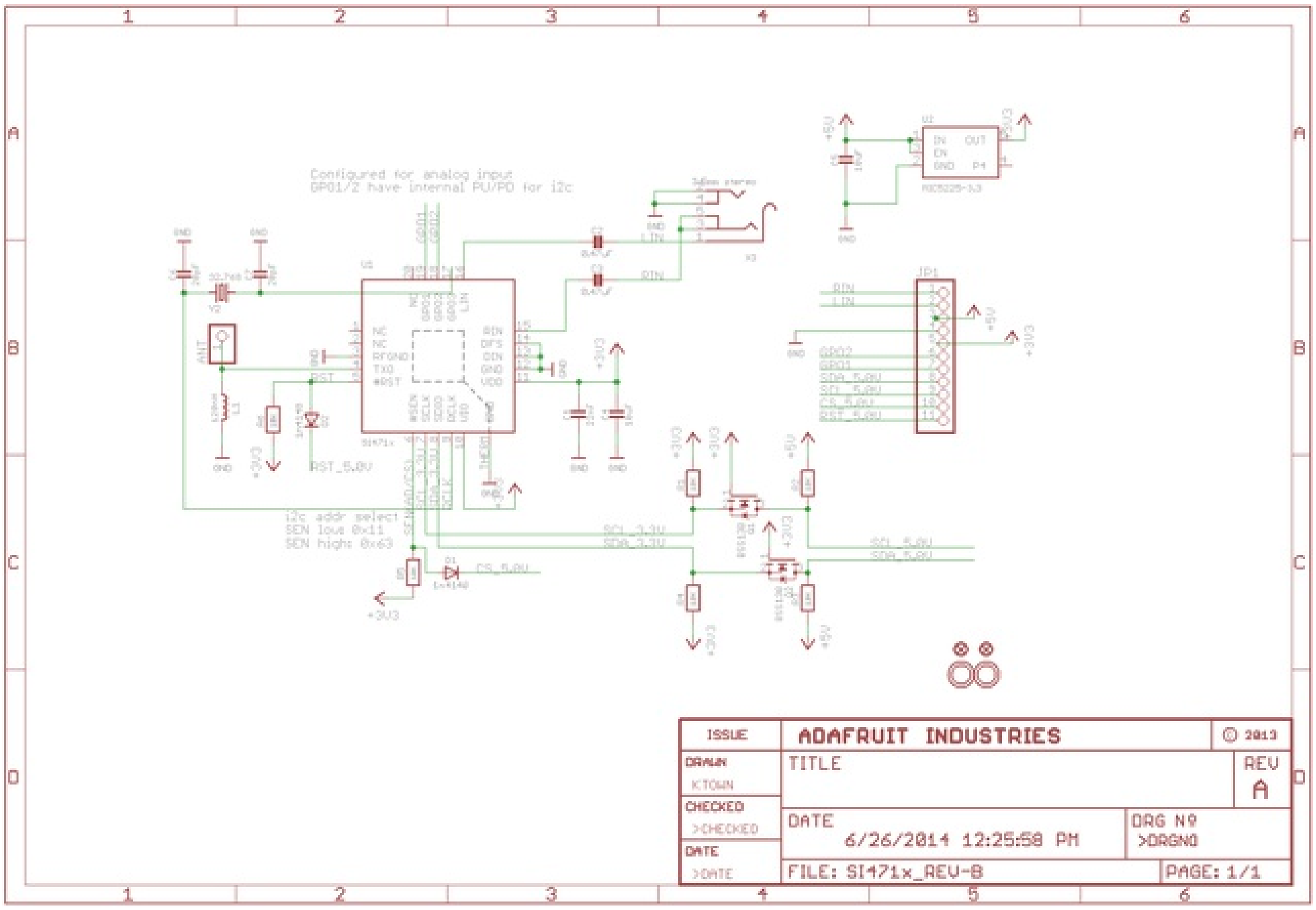
- Si4713 Datasheet (http://adafru.it/dBc)(this does not include any software interfacing details)
- Si47xx Programming guide (http://adafru.it/dBd) - contains all the nitty-gritty details on command data packets etc.

Layout Print

Dimensions in Inches



Schematic


Configured for analog input
GPO1/2 have internal PU/PD for i2c
i2c addr select
SEN low: 0x11
SEN high: 0x63
SI471x
ANT
RST_5.0V
CS_5.0V
SCL_3.3V
SDA_3.3V
SCL_5.0V
SDA_5.0V
JP1
RIN
LIN
+5V
+3V3
GND
ADAFRUIT INDUSTRIES
© 2013
ISSUE
DRAWN
KTOWN
CHECKED
>CHECKED
DATE
>DATE
TITLE
REV
A
DATE 6/26/2014 12:25:58 PM
DRG Nº
>DRGNO
FILE: SI471x_REV-B
PAGE: 1/1