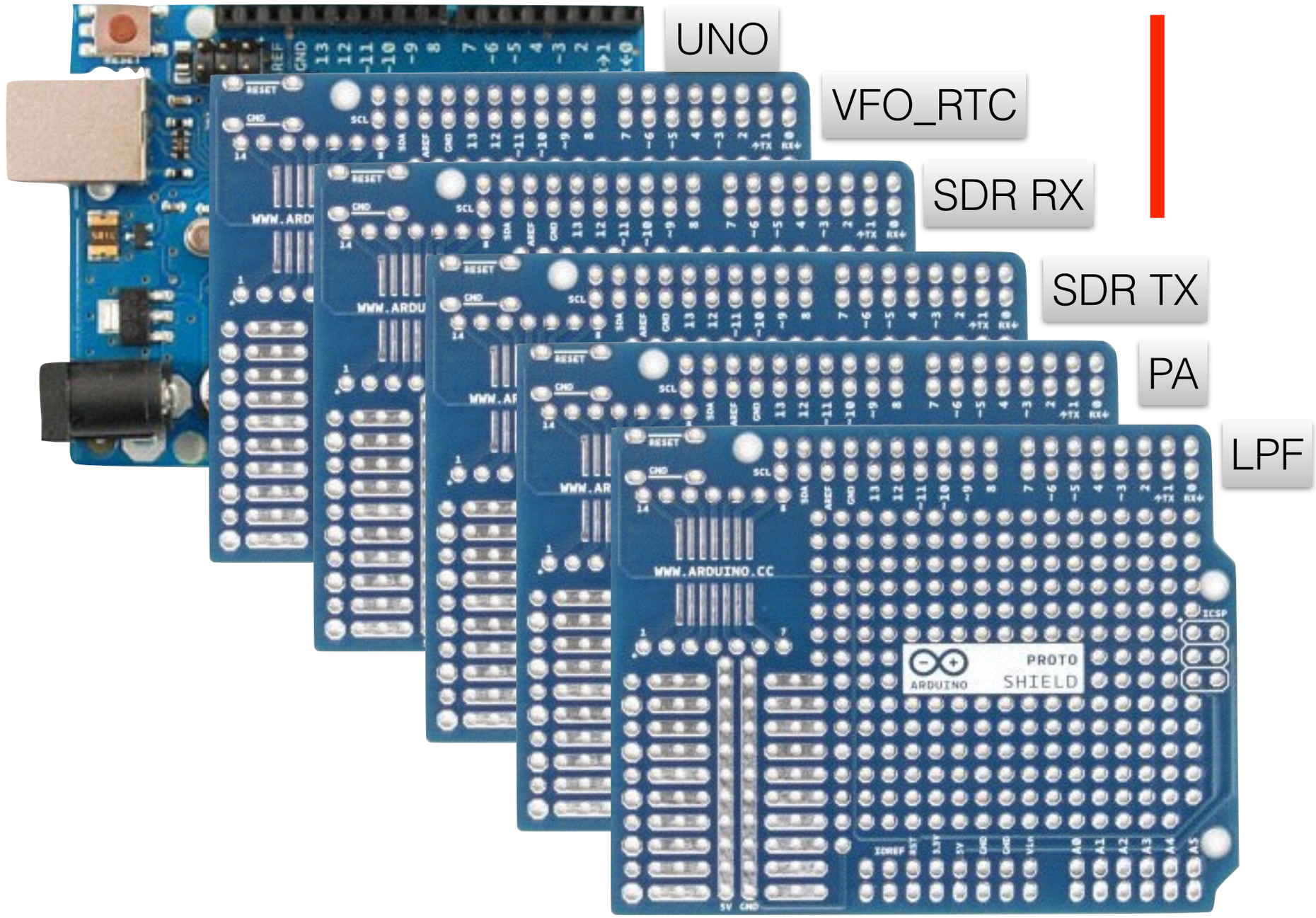
Exciting, learning, sharing

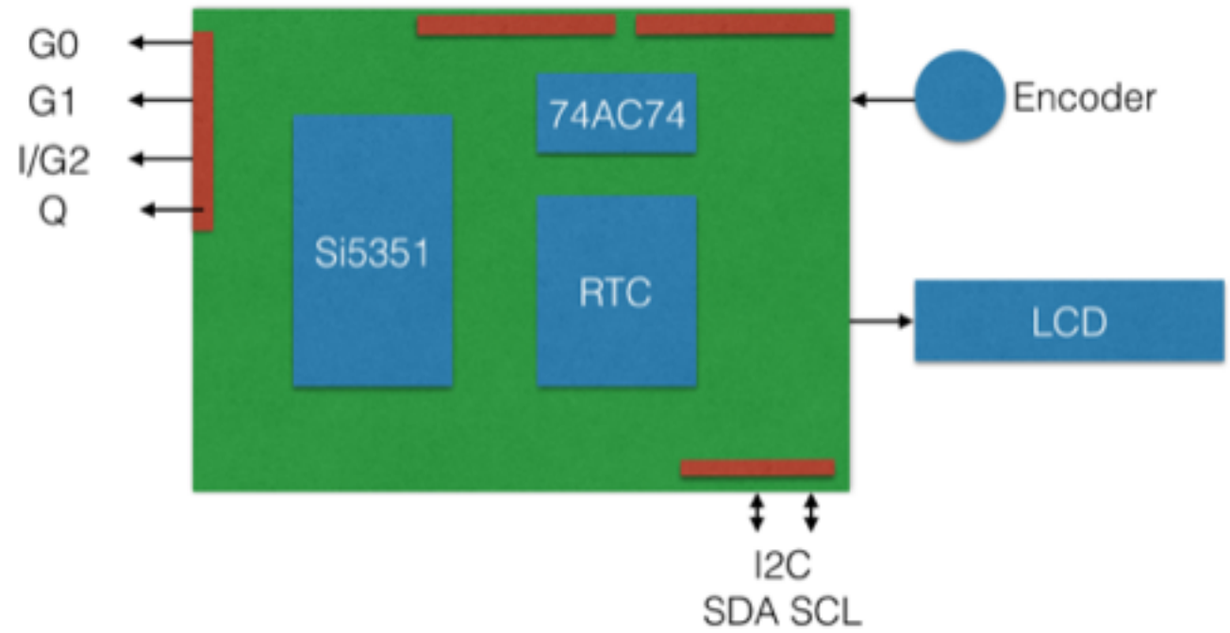
CONCEPT

VFO and RTC

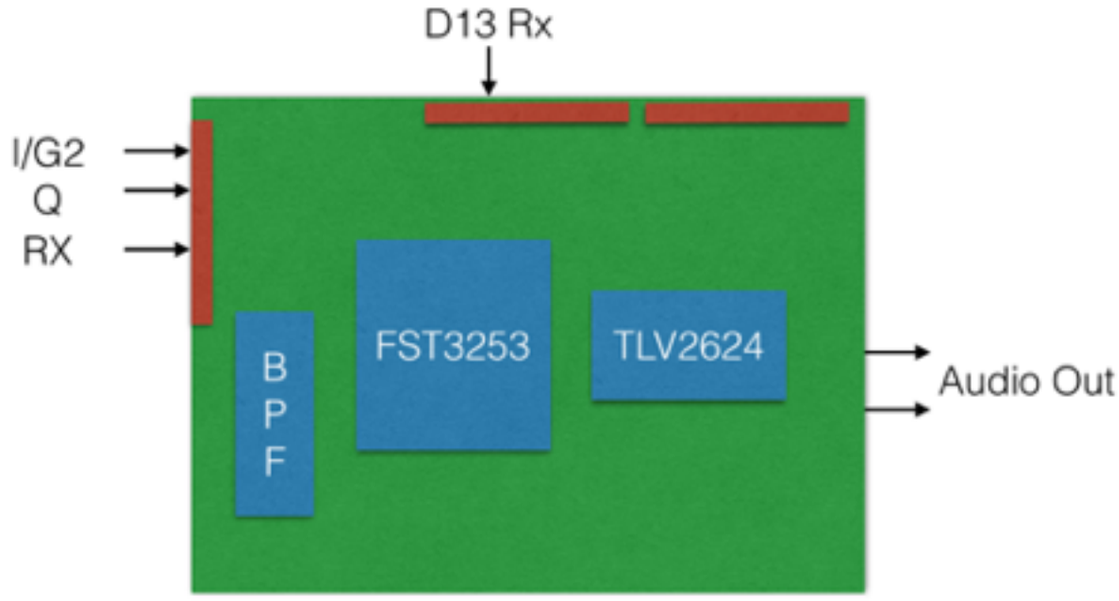
CONCEPT


UNO
VFO_RTC
SDR RX
SDR TX
PA
LPF
WWW.ARDUINO.CC
ARDUINO
PROTO
SHIELD

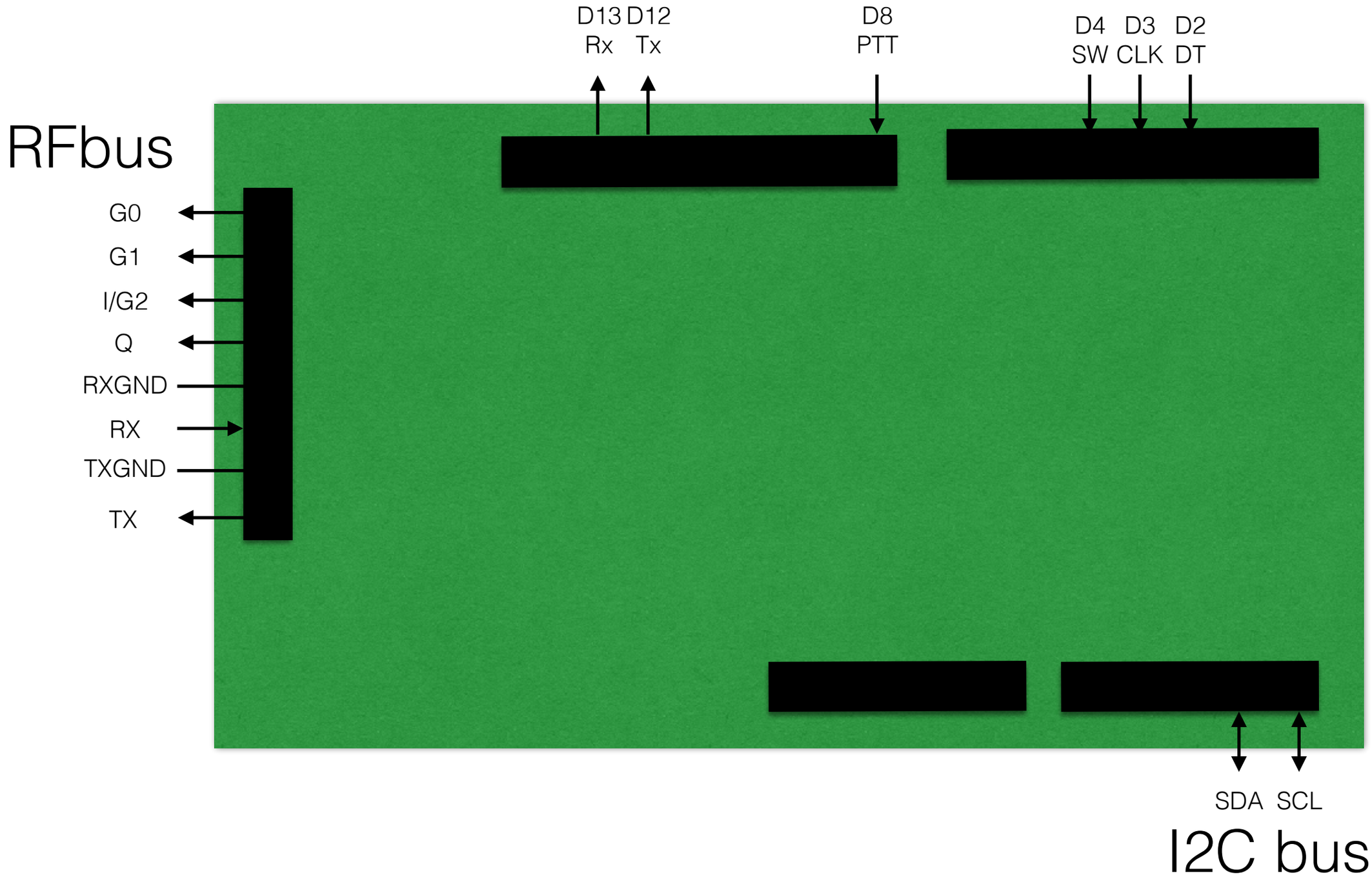
VFO_RTC_IQ



SDRX_40M



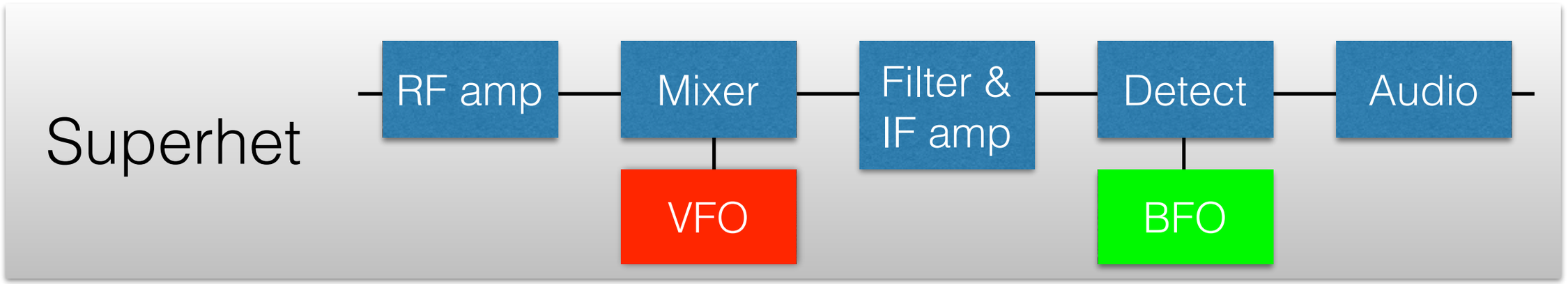
System connections

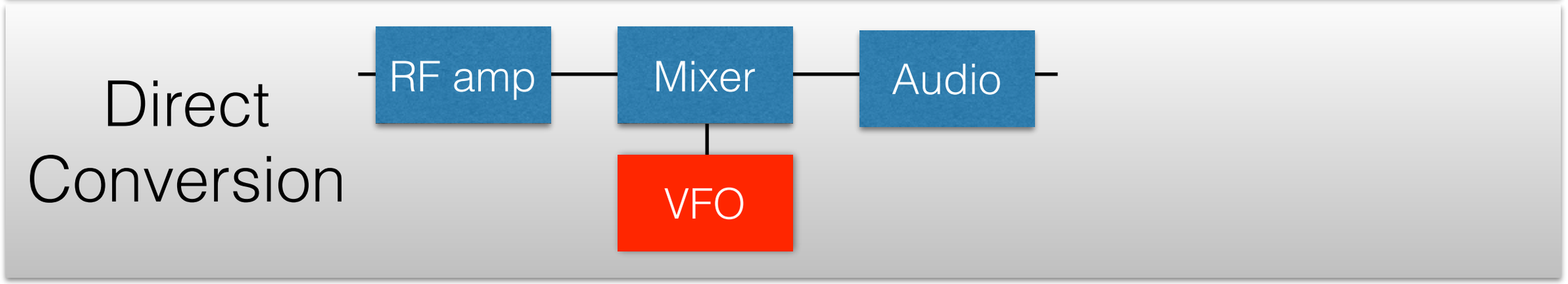


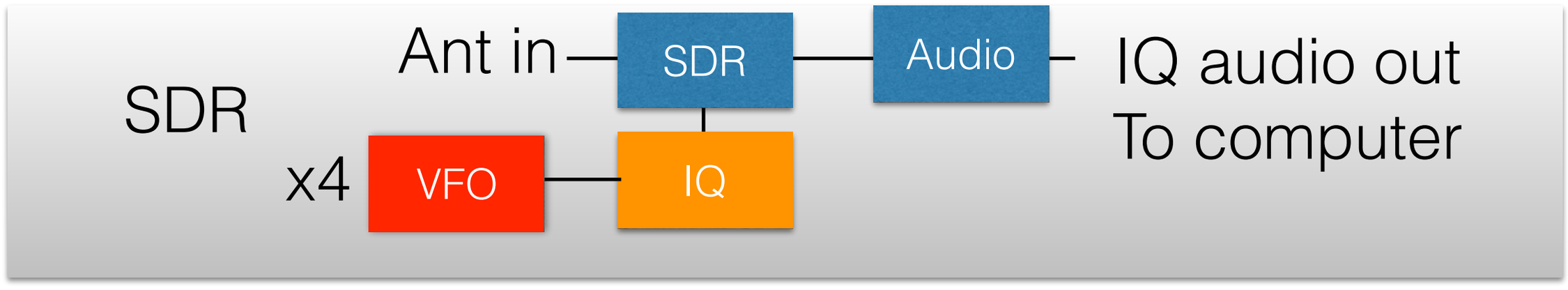
RX & TX architectures

What a “VFO” needs

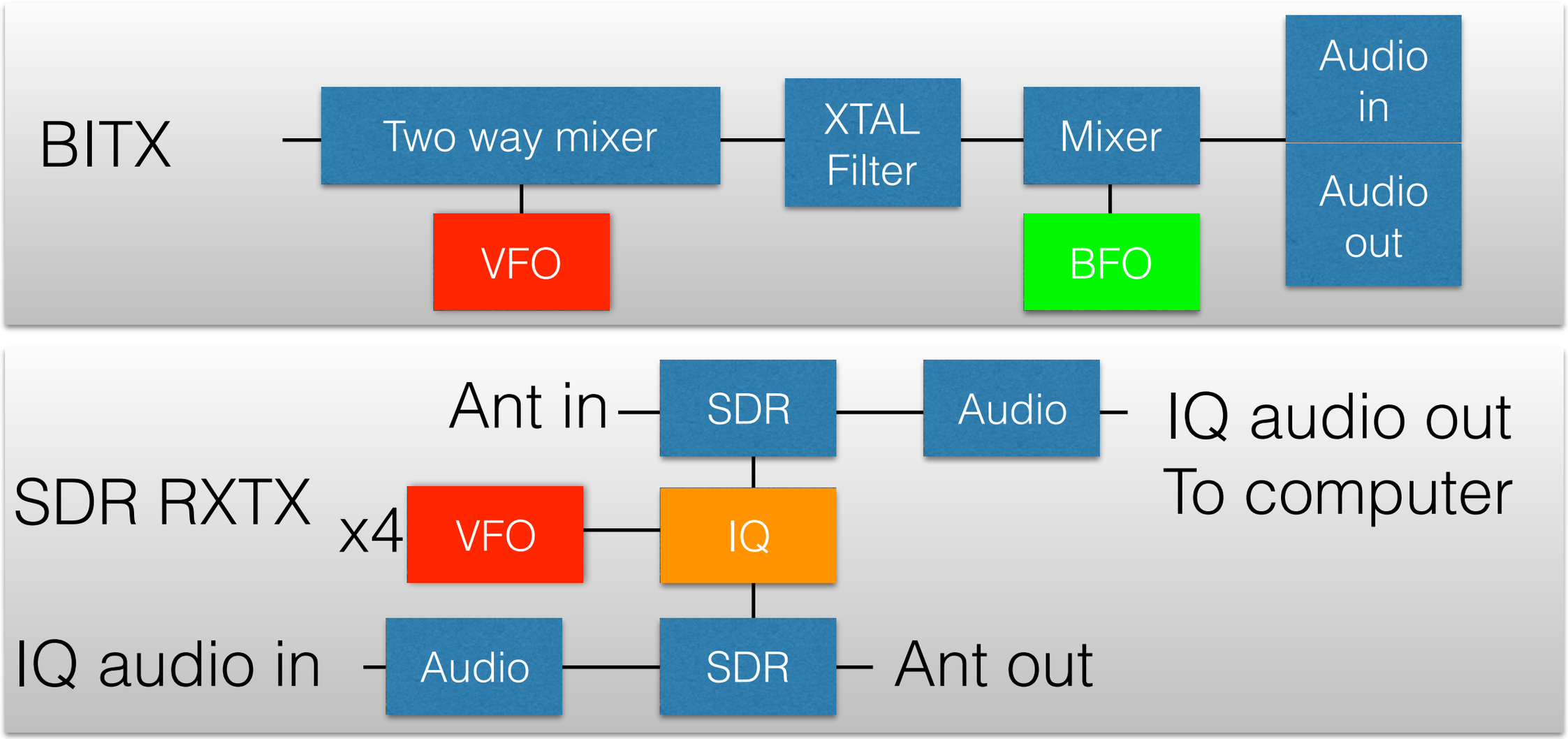
RX architectures

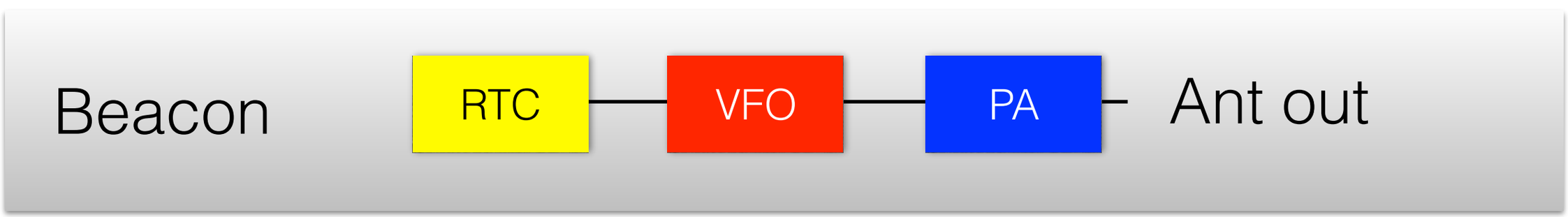




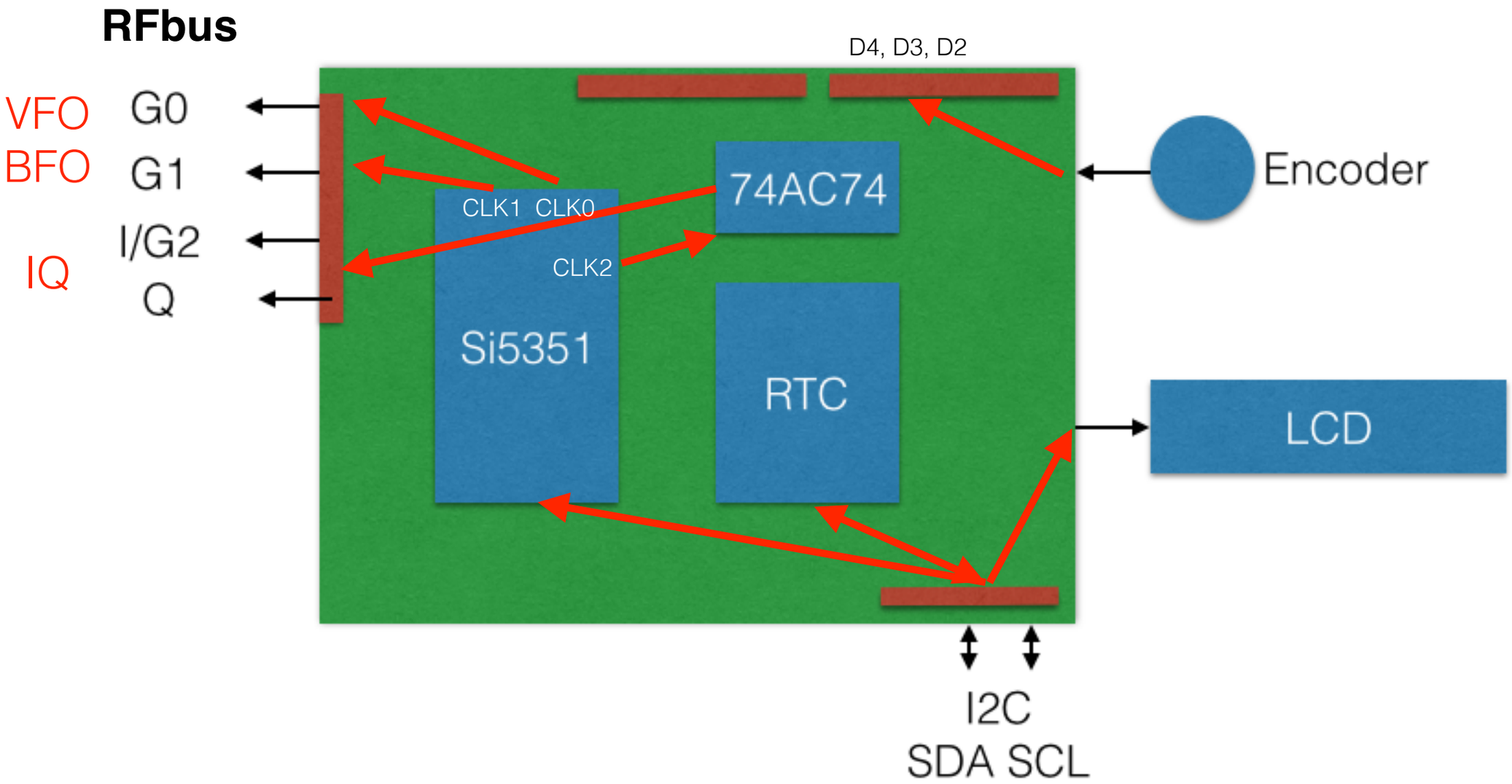


RXTX architectures





The VFO_RTC_IQ



Let’s build a prototype VFO & RTC

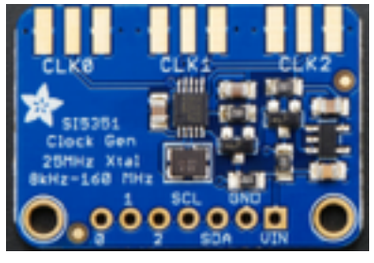
Finally we get to do something!

Components required

Kit 2

Kit 1 & 2

	Starter	Learner	VFO & RTC	VFO+ROT	VFO+ROT+LCD
Arduino UNO	x				
400 point BB	x				
Jumper wires	x				
LED		x			
220R		x			
Piezo buzzer		x			
Si5351 module			x	o	o
Rotary Encoder				x	o
I2C LCD			x		o
RTC module			x		
CR1220 battery			x		
F - M wires			x		o



Missing parts

- When we checked our kits last time we found
  - Some LCD displays were the wrong one, must be I2C version
  - It looked like we had TLV2464 not TLV2462's
  - We were missing the DS3231 RTC modules
- ACTION TAKEN
  - New displays are here, now
  - TLV2462's are here, now
  - RTC modules are on order. Some spares or alternatives are here, now

The Si5351 module

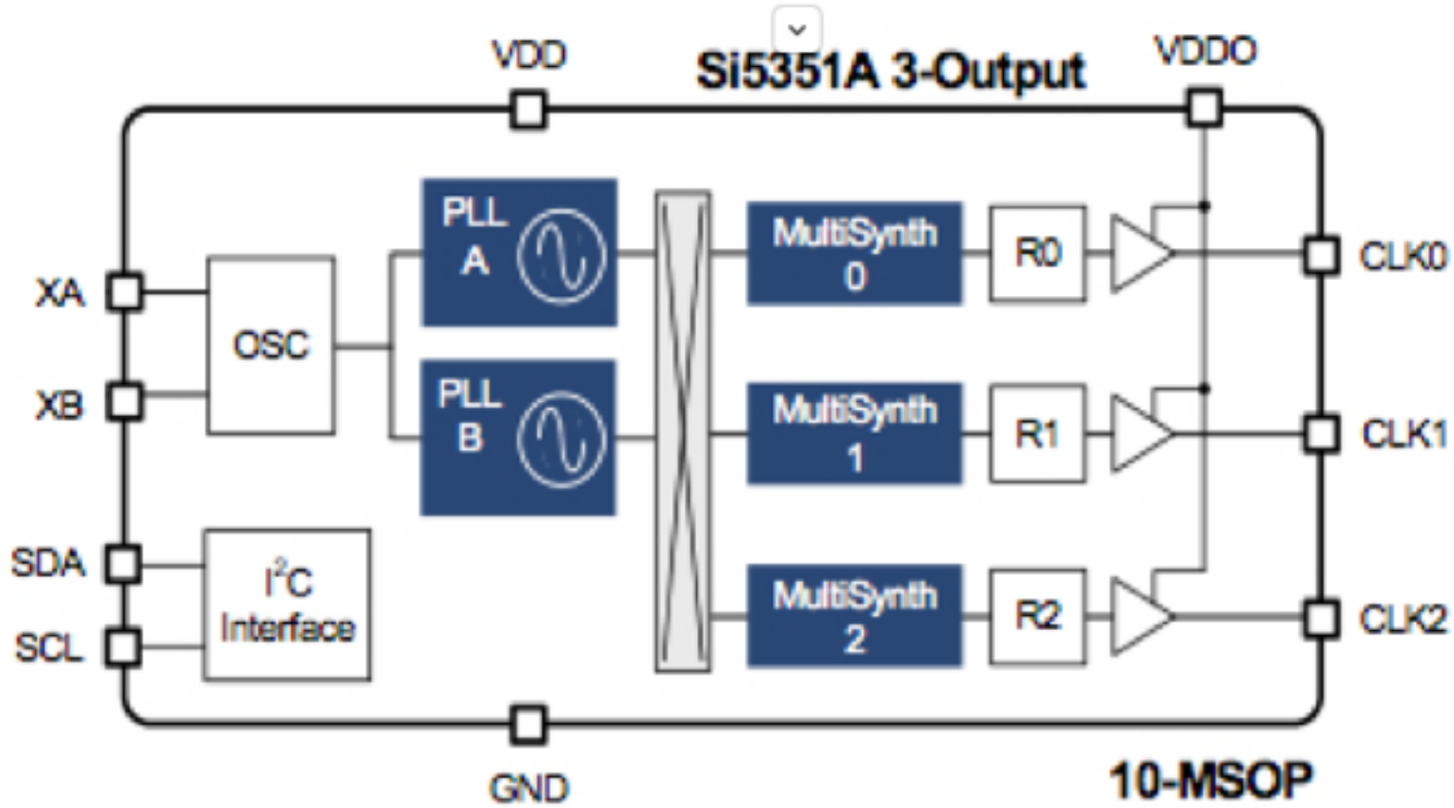
Adafruit Si5351A Clock Generator Breakout Board - 8KHz to 160MHz

- The world moves on
  Si570 —> AD9850 —> Si5351
- Low cost DDS, 3 outputs
- I2C bus controlled
- On-board 5V <-> 3.3V converters
- Simple to use, library by NT7S available on github

etherkit / **Si5351Arduino**

Si5351A - DDS

- Osc at 25MHz
- Two programmed PLLs 600-900MHz
- Switched to three MultiSynths
- And three dividers R0, R1 & R2
- Giving three outputs 8kHz - 160MHz CLK0, 1 & 2
- I2C bus controlled SDA & SCL

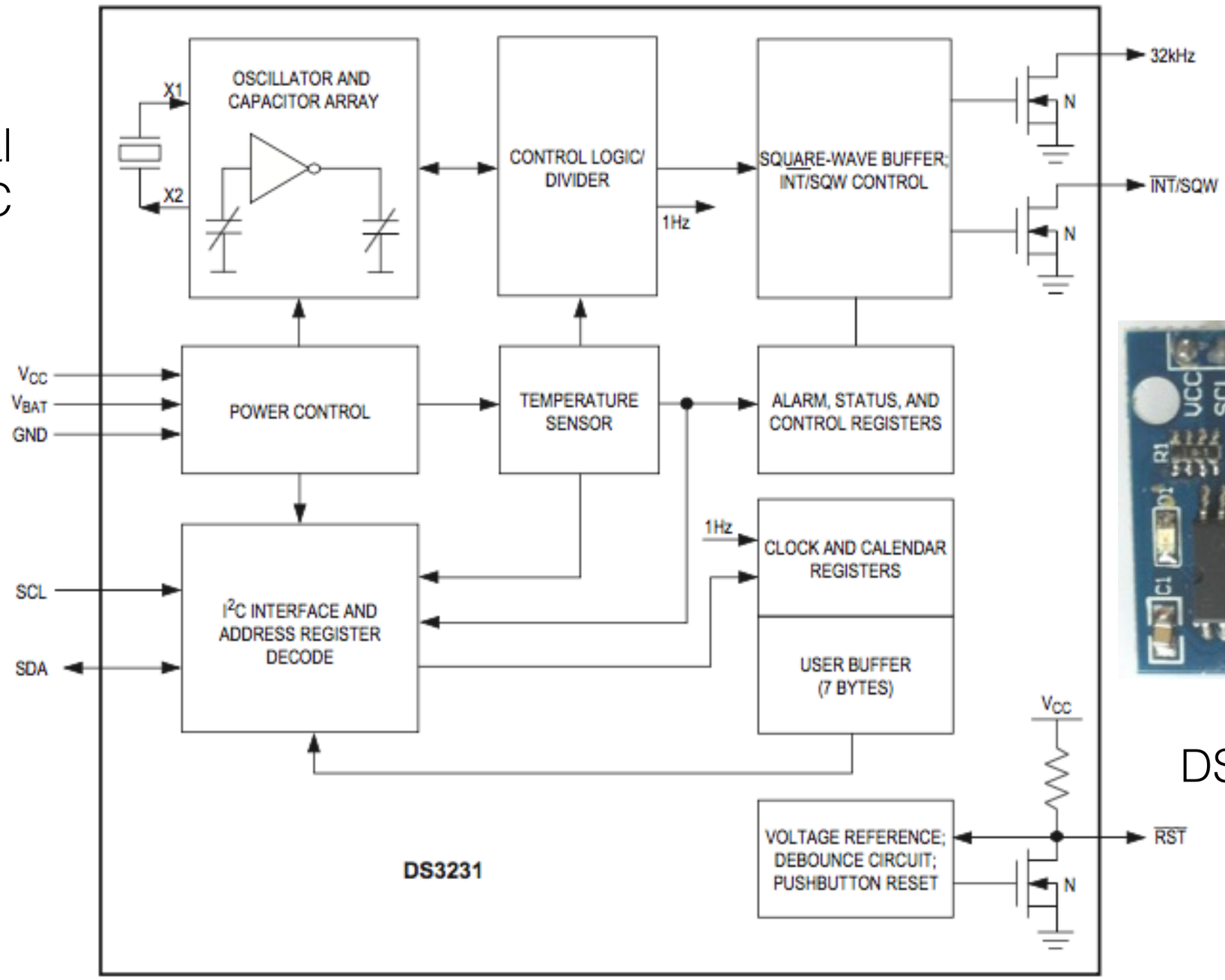


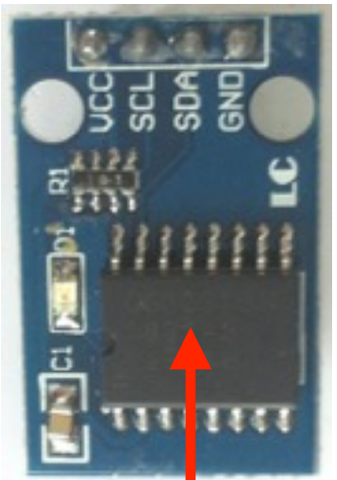
DS3231 - RTC

Internal OSC

2.3-5.5V

I2C bus Controlled

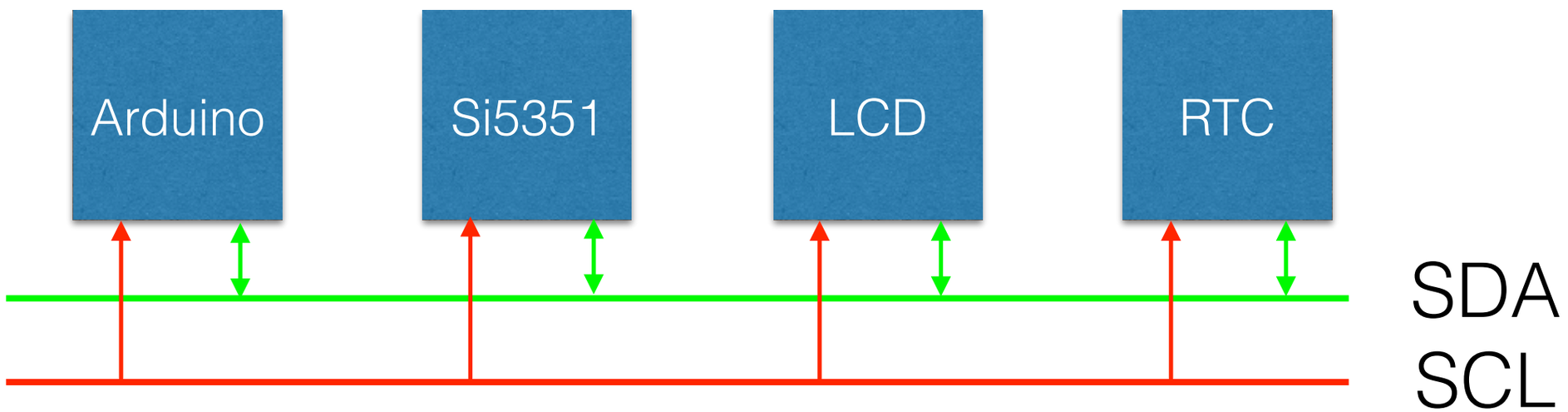




DS3231

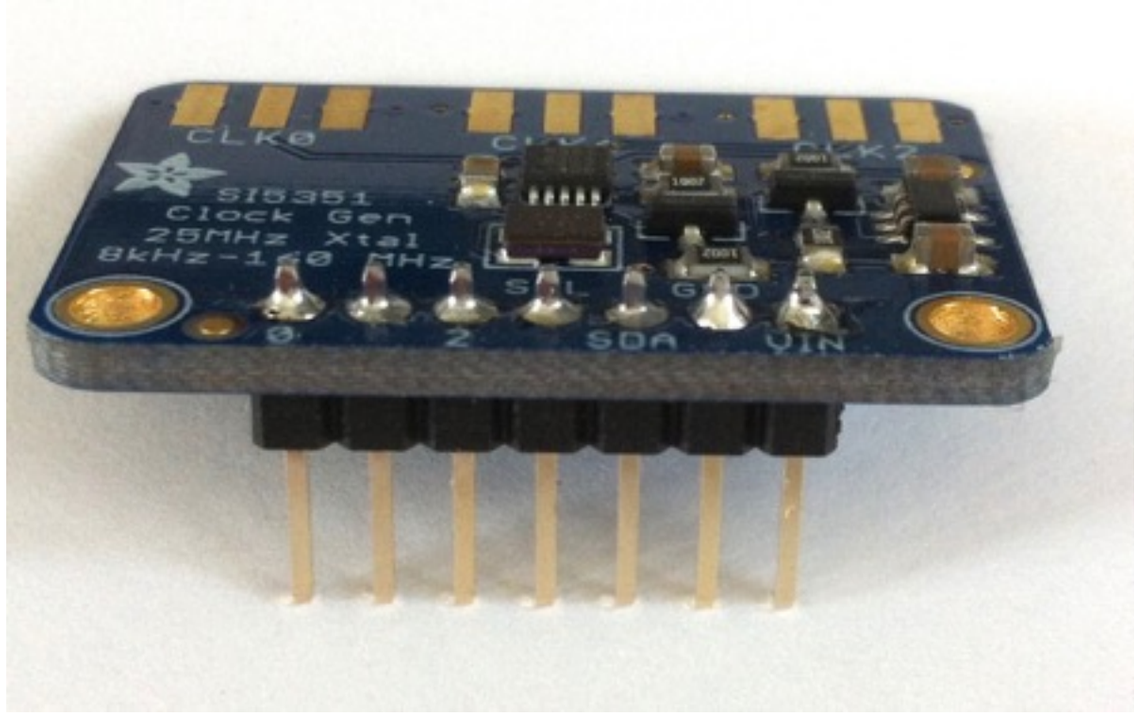
The I2C bus

- The I2C bus was invented over 40 years ago, it was first intended for use in consumer Radio & TV to allow circuits to talk to each other
- It is a 2 wire bus: Clock (SCL) and bi-directional Data (SDA).
- Each circuit connected to the bus has a unique address. So that it can be individually addressed for reading or writing



Homework?

- Did you do your homework?
- Solder the 7 pin header to the Si5351 module



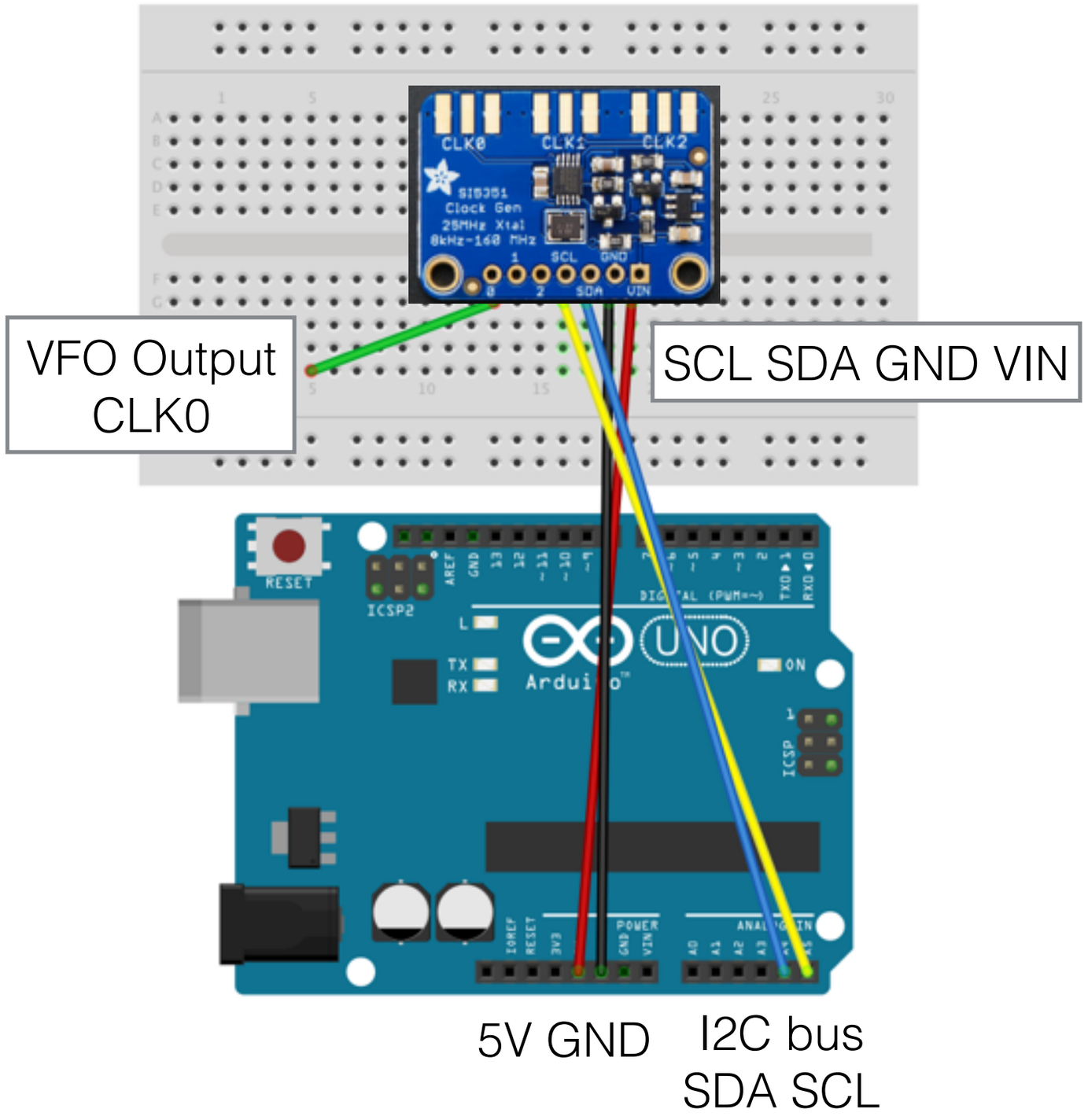
Build a VFO

- Wire up your Si5351
- Take care to get the connections correct

```
SCL - A5
SDA - A4
GND - GND
VIN - 5V
```

- VFO Output is a floating piece of wire, an aerial!

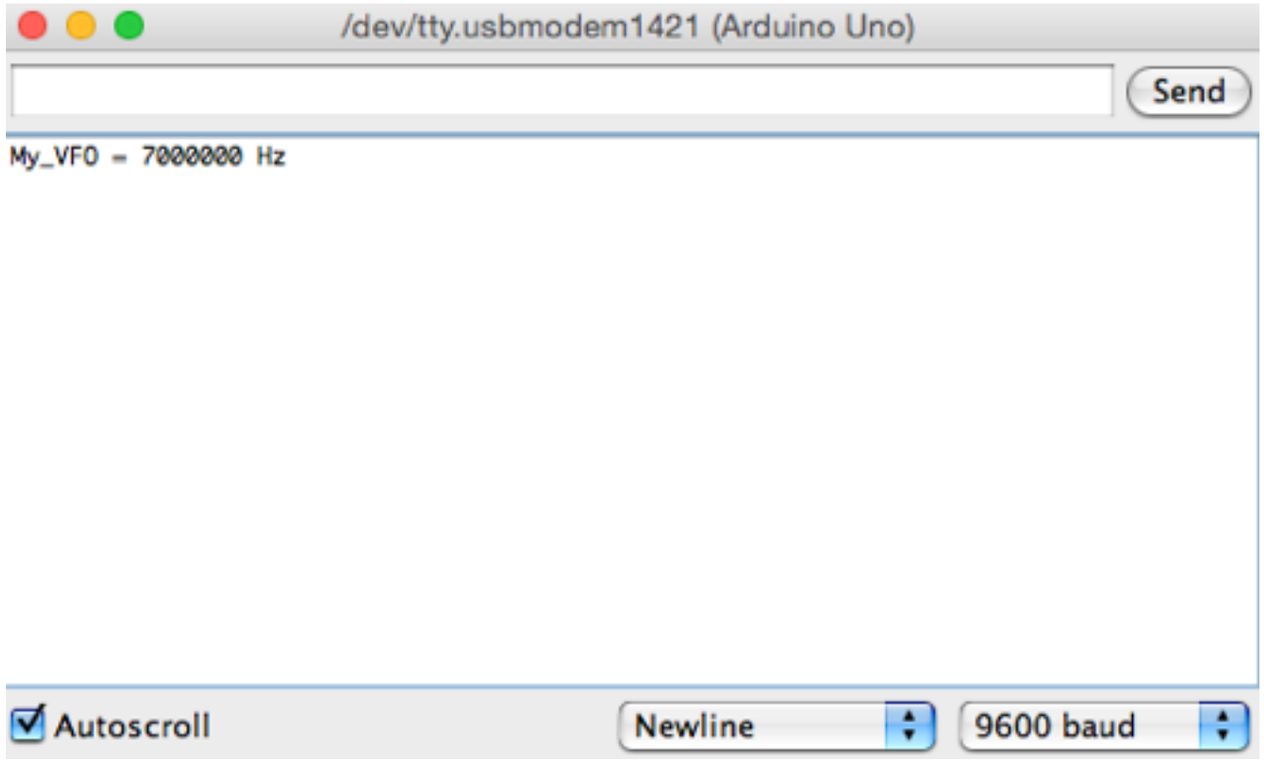
Find out more from the HELP files provided



My_VFO_KB

- File > Sketchbook > My_VFO_KB
- Open the Monitor
- Your VFO will start at 7000000Hz
- Listen for this on a radio
- Enter another frequency in Hz
- Why can't you output 144MHz?

Monitor



Newline 9600

My_VFO_KB

Include libraries I2C & Si5351

Create DDS object

Start up frequency
`uint32_t = unsigned long = 32 bits`

Start serial comms 9600 baud

Initialise the DDS

Enable CLK0 & output freq

```
// I2C and Si5351 Libraries
#include "Wire.h"
#include "si5351.h"

// create dds object
Si5351 dds;

// start frequency (cHz)
uint32_t freq = 700000000; // 7MHz
uint32_t prevFreq = freq;

// setup runs once on upload
void setup(){
  // start serial (monitor "NEWLINE" & 9600 baud)
  Serial.begin(9600);

  // init dds si5351 module, "0" = default 25MHz XTAL
  dds.init(SI5351_CRYSTAL_LOAD_8PF, 0);

  // set 8mA output drive
  dds.drive_strength(SI5351_CLK0, SI5351_DRIVE_8MA);

  // enable VFO output CLK0, disable CLK1 & 2
  dds.output_enable(SI5351_CLK0, 1);
  dds.output_enable(SI5351_CLK1, 0);
  dds.output_enable(SI5351_CLK2, 0);

  freqOut(freq); // output freq
  dispFreq(freq); // display freq in Hz
}
```

A quick look at the code

Get new freq

New?

Output freq, display
Remember previous

Output freq,

```
void loop(){
   freq = getIn(); // get input freq cHz

  // new freq?
  if(freq != prevFreq)
  {
    freqOut(freq); // output freq
    dispFreq(freq); // display in Hz
    prevFreq = freq; // remember as previous freq
  }
}

// freq output in cHz on CLK0
void freqOut(uint32_t freq){
    dds.set_freq(freq, 0, SI5351_CLK0); // cHz
}
```

Note: Si5351 library
uses frequency in 0.01Hz steps (cHz)

A quick look at the code

Get new freq in

Display freq

```
// get input Hz, return cHz
uint32_t getIn(){
  uint32_t in;

  while(Serial.available() > 0) // flush buffer
    Serial.read();

  while(Serial.available() == 0) // wait for input
    in = Serial.parseInt(); // read input in Hz and parse to integer

  return in * 100; // return in cHz
}

// display frequency on monitor
void dispFreq(uint32_t f){
  // display freq
  Serial.print("My_VFO = ");
  Serial.print((float)f / 100 , 0); // convert to float & display in Hz
  Serial.println(" Hz");
}
```

Sketch does NOT work above ~43MHz…. why?

Why not 144MHz?

```
8  bits unsigned byte          uint8_t   0 - 255

16 bits unsigned int           uint16_t  0 - 65,535

32 bits unsigned long          uint32_t  0 - 4,294,967,295

     so 4,294,967,295 cHz = ~43MHz max

64 bits easily handles freq up to max of Si5351, 160MHz
```

- Change **`uint32-t`** to **`uint64_t`** in sketch
  re-upload and and enter 144MHz
- Check on a VHF radio

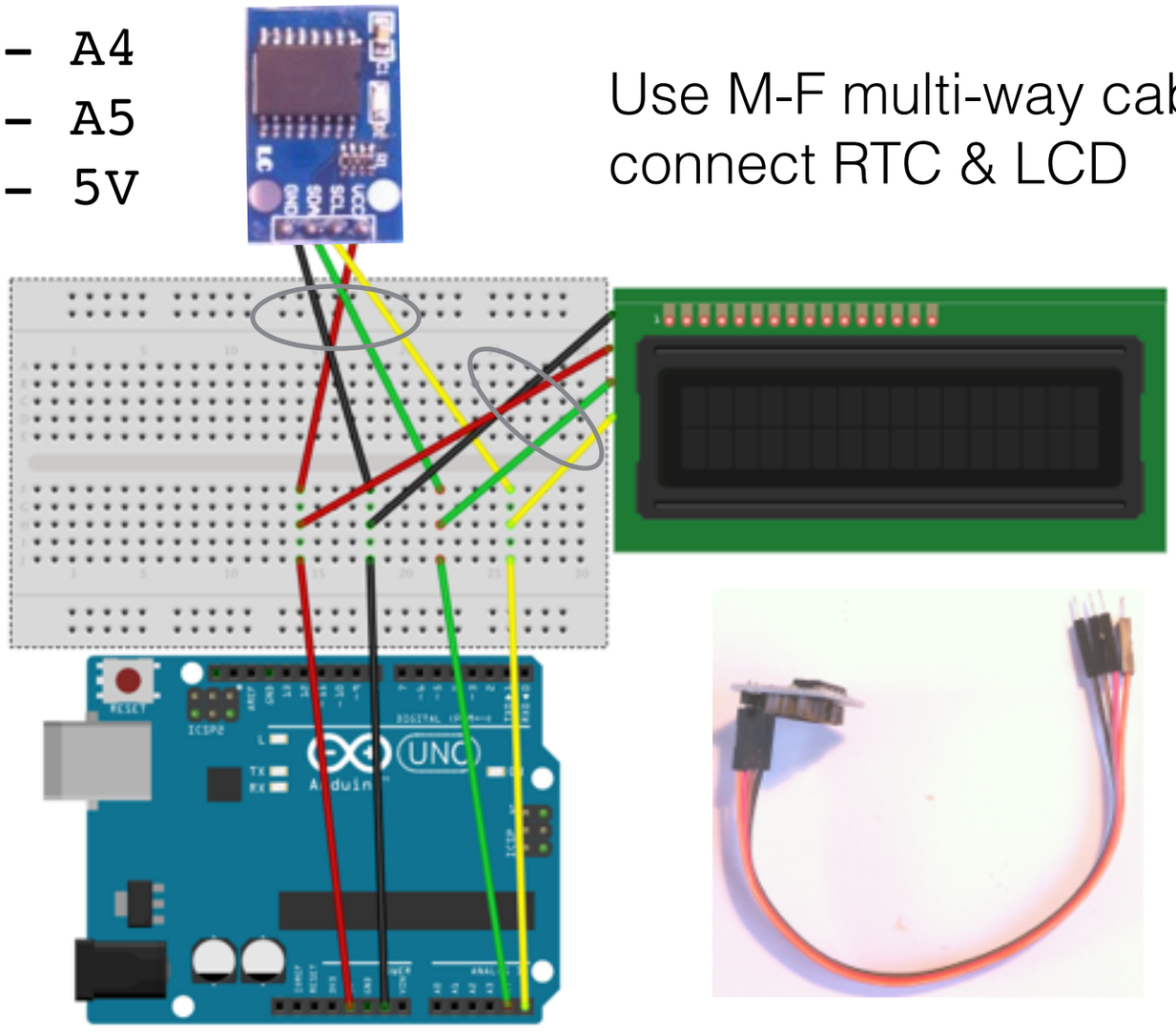
- Could you make an FM transmitter?
- Could you make a 144MHz beacon TX?

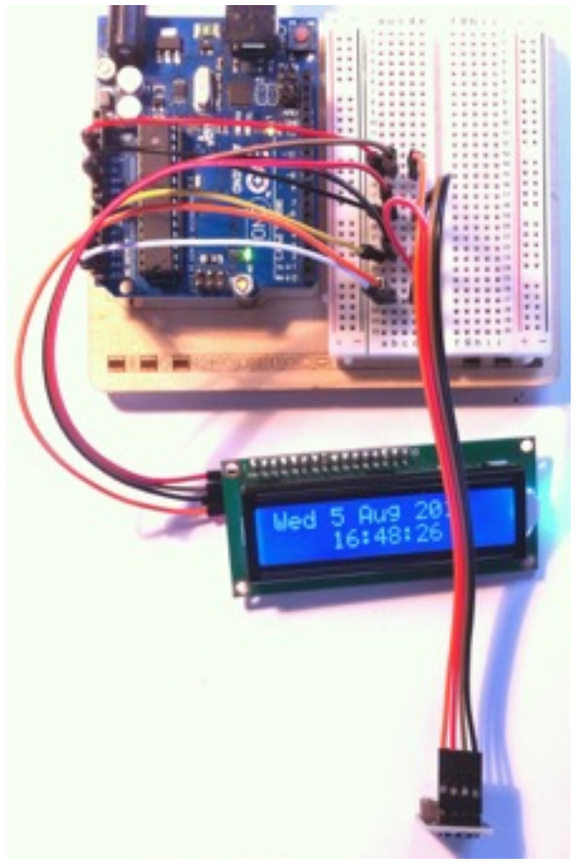
Build an RTC

Insert the CR1220 battery
in the back with +ve up

Use M-F multi-way cables to
connect RTC & LCD

```
GND - GND
SDA - A4
SCL - A5
VCC - 5V
```





+5V GND SDA SCL

Set your clock

- When the RTC is first powered up the date & time are ***not*** set
- To set the date & time, a special sketch sets them from numbers you enter on the Monitor
- File > Sketchbook > My_ RTC_set
- Enter the date & time on the monitor, press SEND to set RTC
  - Today: `1509152193000 = 2015 Sept 15 Tues 19:30:00`
  - `YY = 15, MM = 01 Jan, DD = 01-31, w = 1 Mon,`
    `HH = 00-23hr, MM = 00-59, SS = 00-59`
- You can repeat (re-upload sketch) to re-set the date/time again

My_RTC_LCD

- File > Sketchbook > My_RTC_LCD
- This sketch reads the RTC and displays the date & time on your LCD display
- It is different to the “My_ RTC_set” program as it does ***not*** use the DS3231 library but reads the RTC registers directly which is faster

  We need the clock read to be fast so that time in the program “loop” is given to detecting encoder turns…

- Can any clever person write some code to calibrate the clock using the Rotary Encoder to set the seconds exactly? Or use a GPS RX?

My_RTC_LCD

```
// libraries
#include "Wire.h"
#include "LiquidCrystal_I2C.h"

// RTC I2C address
#define RTCADDR 0x68

// LCD
#define LCDADDR 0x27
#define LCDCOLS 16
#define LCDROWS 2

// LCD object
LiquidCrystal_I2C lcd(LCDADDR, LCDCOLS, LCDROWS);

// RTC time and date
byte Second, prevSecond, Minute, Hour, DoW, Date, prevDate, Month, Year;
```

```
void setup() {
  // initialise the wire library for I2C comms
  Wire.begin();

  // init LCD & backlight on
  lcd.init();
  lcd.backlight();

  getRTC(); // get time
  dispDate(0, 0); //  display date & time
  dispTime(4, 1);
  prevSecond = Second; // save current second & date
  prevDate = Date;
}

void loop() {
  getRTC(); // get time
  if (Second != prevSecond) {
    dispTime(4, 1); // display it, if changed
    prevSecond = Second;
  }
  if (Date != prevDate) {
    dispDate(0, 0);
    prevDate = Date;
  }
}
```

```
// get time from RTC, convert bcd to decimal
void getRTC() {
  // Reset the RTC register pointer
  Wire.beginTransmission(RTCADDR);
  byte zero = 0x00;
  Wire.write(zero);
  Wire.endTransmission();

  // request 7 bytes from the RTC address
  Wire.requestFrom(RTCADDR, 7);

  // get the time data
  Second = bcdToDec(Wire.read()); // 0 - 59
  Minute = bcdToDec(Wire.read()); // 0 - 59
  Hour = bcdToDec(Wire.read() & 0b111111); // mask 12/24 bit
  DoW = bcdToDec(Wire.read()); //0 - 6 = Sunday - Saturday
  Date = bcdToDec(Wire.read()); // 1 - 31
  Month = bcdToDec(Wire.read()); // 0 = jan
  Year = bcdToDec(Wire.read()); // 20xx
}

// Convert binary coded decimal to normal decimal numbers
byte bcdToDec(byte val) {
  return ( (val / 16 * 10) + (val % 16) );
}
```

bcd		dec
0001	0011 ->	00001101
1	3	13

Display Functions

- Now have a look yourself at the display functions, note the use of the **`switch case`** statements
  - **`void dispdate(byte c, byte r)`** - prints day of week, month and year at col/row
  - **`void disptime(bytec, byte r)`** - prints time at col/row

Now you have no excuse.
You can't be late for S3

Homework - study the code, time an egg?

Next time

Design a PCB
using
Eagle