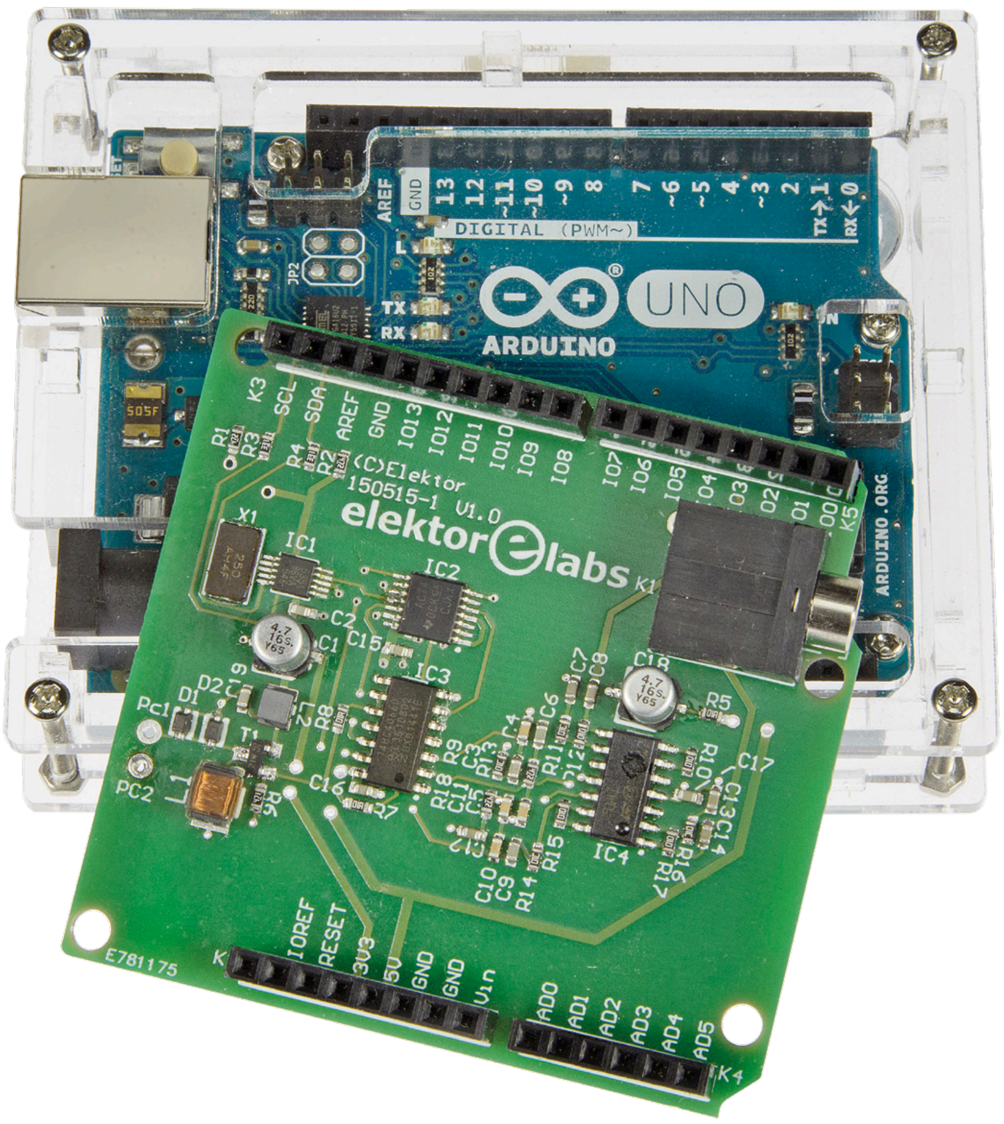
Elektor SDR Reloaded

SDR Shield for the Arduino

By **Burkhard Kainka** (Germany)



A Software Defined Radio is a universal tool in RF technology circles, one that can also be put to use for making measurements. The characteristics of the receiver are defined in software, which now gives us the opportunity to use an Arduino Shield as a front-end.

Technical Characteristics

- Supply voltage: 5 V and 3.3 V as for Arduino
- Frequency range: 150 kHz up to 30 MHz
- Sensitivity: 1 µV
- Total amplification: 40 dB
- Maximum antenna signal level: 10 mV
- Dynamic range: 80 dB

Even though broadcast services are deserting the AM domains in the long, medium and short wavebands, there is still plenty of interest to be found surfing the radio waves with a home-constructed receiver. Now more than ever you might say, because many distant stations now come up far more clearly because they are no longer swamped by stronger signals. In fact it is often so quiet on the short waves that it’s easy to imagine your receiver has gone deaf. On some bands it is the radio amateurs who produce the strongest signals. And there is always something new to find, from pirate radio stations through SSB radiotelephony to the new digital modes. That just has to make you curious!

Elektor has already published many radio and receiver projects. A Software Defined Radio with USB interface was introduced as long ago as 2007 [1]. In the meantime much thought has been devoted to conceiving updates for this design. However, the PLL chip we used originally is no longer made, making it necessary to find a new solution. This has arrived in the form of the Silicon Lab SI5351 chip, a CMOS clock generator from 8 kHz to 160 MHz with I^{2}C bus.

First investigations revolved around a break-out board from Adafruit. The available sample software was written for the Arduino, so our first steps were undertaken with the Arduino. The new VFO was simply hooked up to the old SDR PCB and proved its suitability (**Figure 1**).

And then there came an idea: why not simply build the entire receiver as an Arduino Shield? This decided the power supply requirements, using the USB interface already available on the PC. The Arduino would look after controlling the VFO and could be addressed in plain language so to speak (6030 kHz please). And what is perhaps even more exciting, this even gives you a real chance to build a totally standalone receiver. Operation could be migrated from the PC to the Arduino relatively simply. And who knows, perhaps one day the decoding of the IQ signal as well?

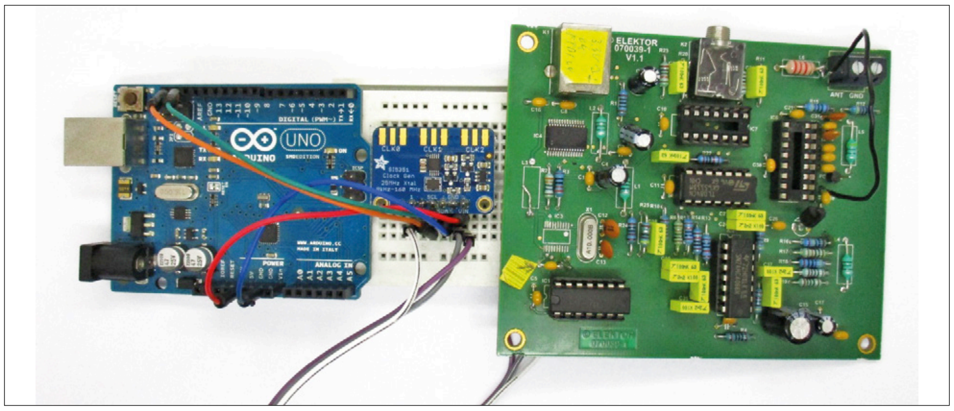


Figure 1. First preliminary test for an SDR2: an SI5351 PLL chip hooked up to an Arduino Uno and the 'old' SDR receiver.

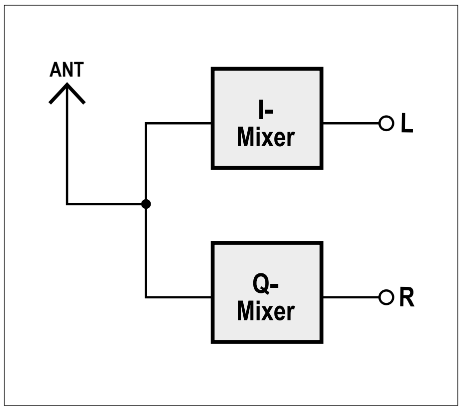


Figure 2. Working principles: the front-end consists of a dual direct mixer with signals shifted 90 degrees out of phase.

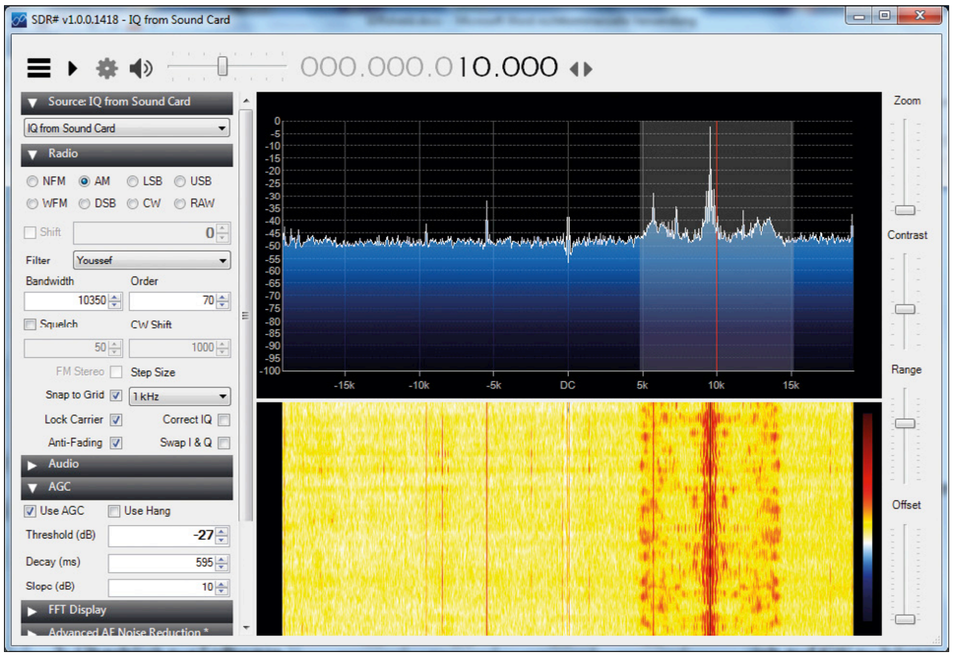
So how does it work?

First, let's go back to basics. What fundamentally is a Software Defined Radio? For quite some time the development of digital electronics left radios entirely untouched. When home computers first became available, most radios were still analog. Then development began to take place, at least for digitizing their tuning. Today's radios are often equipped with a PLL synthesizer that simplifies tuning and guarantees precise conformity to channel spacing. The rest of the circuitry remains analog as previously.

Subsequently digital electronics appeared inside commercial RF equipment and amateur radio gear. Ever more of the analog functions in devices were replaced by software. In most cases a digital signal processor (DSP) with appropriate software operates out of sight from the user and takes care of optimal filter curves, variable bandwidth, signal decoding, interference suppression and much more. The equipment is altogether improved and with less hardware overhead. Further examples of this kind of development can be found in smartphones and other portable end-devices. At the same time it's evident that hobby constructors can no longer keep pace with this technology. In fact things don't need to be so sophisticated, however. All you need is a rapid A-D converter connected direct to the antenna. The entire spectrum is digitalized and then further processed digitally. In fact technology of this kind is available for the entire frequency range from 0 to 30 MHz. It's software alone that filters out specified frequencies and demodulates the desired signal. Regardless of whether we are dealing with AM or DRM broadcast stations or whether we're receiving SSB signals, CW Morse transmissions, teletype (RTTY), weather fax or whatever else, everything is feasible. There is appropriate software for everything. Sure, it must be conceded that with such a large bandwidth, the hardware can be quite expensive, and the further processing necessary for this broad spectrum imposes high demands.

Nevertheless a cunning way around this challenge can be found in the sound cards of modern PCs. Using the 96 kHz sampling rate that's normal today you can already receive the whole frequency range up to 48 kHz. Instead of using a microphone, you simply connect a large coil as an antenna and immediately you're able to receive the VLF band. Down there are plenty of interesting signals, even transmitters installed on submarines.

If you want to use the sound card for higher frequencies, you must first down-convert the signals. The process is akin to a superhet with a lower intermediate frequency (IF). The PC handles the IF stages, filtering, automatic gain control (AGC) and demodulation. In principle a simple direct mixer with a diode ring mixer or the well-known NE612 would be adequate for this. Only a stable variable oscillator (VFO) would be needed in addition. For special applications you could


SDR# v1.0.0.1418 - IQ from Sound Card

000.000.010.000

Source: IQ from Sound Card
IQ from Sound Card

Radio
NFM [✓ AM] LSB USB
WFM DSB CW RAW
Shift 0
Filter Youssef
Bandwidth 10350 Order 70
Squelch 50 CW Shift 1000
FM Stereo Step Size
Snap to Grid ✓ 1 kHz
Lock Carrier ✓ Correct IQ
Anti-Fading ✓ Swap I & Q

Audio

AGC
Use AGC ✓ Use Hang
Threshold (dB) -27
Decay (ms) 595
Slope (dB) 10

FFT Display
Advanced AF Noise Reduction *

Figure 3. The SDR# program receiving an AM signal.

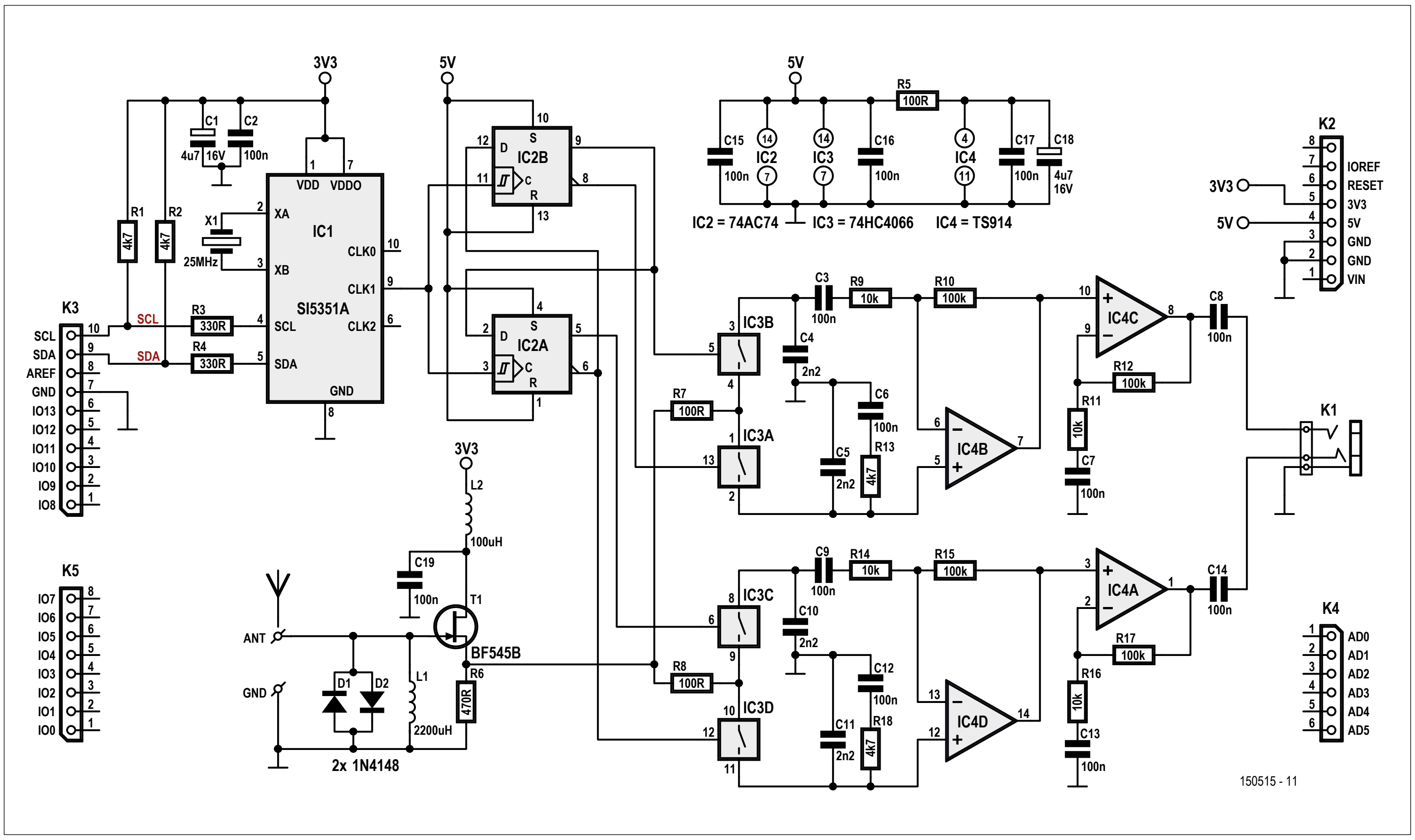


Figure 4. Schematic of the new SDR receiver.

use a crystal oscillator. But if you want to be able to tune an entire band, it ought to be a DDS generator or a PLL module. Simply stated, an IQ mixer is a matter of a dual direct mixer with two signals phase-shifted by 90 degrees. The oscillator signal is always adjacent to the reception frequency. The output signals therefore lie within the AF range, mostly between 0 kHz and 24 kHz. The two signals are designated I and Q (see **Figure 2**). These are applied direct to the

Component List

Resistors
R1,R2,R13,R18 = 4.7kΩ 1%, 0.1W, SMD 0603
R3,R4 = 330Ω 1%, 0.1W, SMD 0603
R5,R7,R8 = 100Ω 1%, 0.1W, SMD 0603
R6 = 470 Ω 1%, 0.1W, SMD 0603
R9,R11,R14,R16 = 10kΩ 1%, 0.1W, SMD 0603
R10,R12,R15,R17 = 100kΩ, 0.1W, SMD 0603

Capacitors
C1,C18 = 4.7µF 16V, SMD case B
C2,C3,C6,C7,C8,C9,C12,C13,C14,C15,C16,C17,C19 = 100nF 50V, X7R, SMD 0603
C4,C5,C10,C11 = 2.2nF 50V, X7R, SMD 0603

Inductors
L1 = 2200µH (Fastron L-1812AF)
L2 = 100µH (Murata LQH32CN101K23L)

Semiconductors
D1,D2 = 1N4148WS, SOD-323
T1 = BF545B, SOT-23
IC1 = SI5351A-B-GT, MSOP-10
IC2 = SN74AC74PW, TSSOP-14
IC3 = 74HC4066, SOIC-14
IC4 = TI914IDT, SOIC-14

Miscellaneous
K1 = stereo jack socket, 3.5mm, PCB mount
K2,K3,K4,K5 = connector set, Arduino compatible (1 pc. 6-pin, 2 pcs. 8-pin, 1 pc. 10-pin)
X1 = 25MHz quartz crystal (Abracon ABM7)

PCB # 150515-1
Or
PCB with preassembled SMD parts: 150515-91

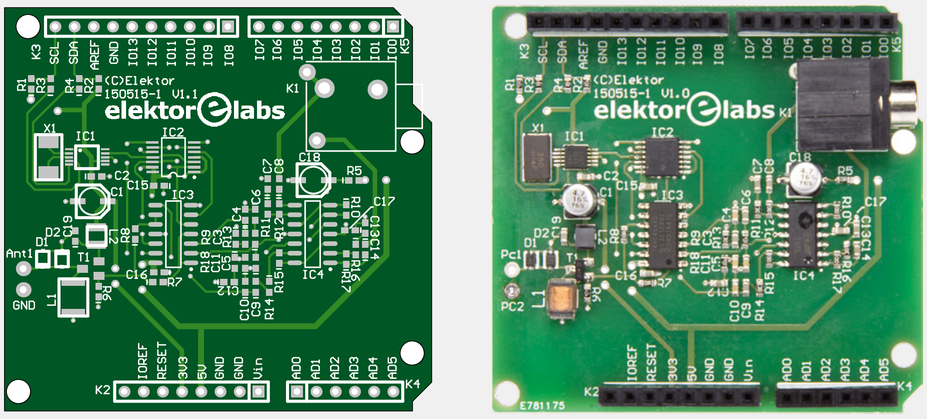


Figure 5. Double-sided PCB for the SDR3 executed as an Arduino Shield.

left and right channels of the sound card input. The rest is dealt with in software. A simple mixer would mix the range below and above the oscillator frequency into the same range, in which the dreaded image frequency problem would arise. By carrying out dual mixing and phase shifting, the software is able to cancel out and eliminate the image frequency, however. In this way a range between –24 kHz and +24 kHz can be received if the sound card has a sampling rate of 48 kHz. **Figure 3** shows what a program like SDR# makes out of this (see also the text panel *SDR software*).

Circuit

A glance at the schematic in **Figure 4** shows the individual building blocks. The SI5351 PLL generator (IC1) delivers the oscillator signal with the 4x receive frequency to the 74AC74 divider (IC2B). This divides the frequency by four and delivers the signals phase-shifted by 90 degrees to the 74HC4066 mixer (IC3). This analog switch is wired as a changeover switch and applies the RF signal alternately to the inverting and non-inverting inputs of the TS914 op-amp (IC4B/IC4D). In this way the signal is mixed down into the AF region. After some modest filtering and amplification (IC4C/IC4A), the signal reaches the audio output. The RF input stage creates a source follower using the BF545B JFET (T1), the SMD equivalent of the BF245B. Anyone familiar with the old Elektor SDR will see a certain simplification in the signal path. On its RF input it had several switchable lowpass filters. The new design has a wideband input and is protected against over-voltage by two diodes. This is completely adequate for shortwave reception with a wire antenna. The additional over-voltage protection was born from experience with the first SDR; in a thunderstorm the input stages could be damaged. For specific purposes you can also use additional external filters and a preamp. In the old version the AF amplification could be adjusted in three steps. This time around there is only the middle level, which proved itself to be fine for general use. So everything has become a bit simpler and now works well with the Shield.

For first trials you simply need to connect up a wire antenna. Some wire with a length of three meters hung from the ceiling will be fine. If this is impossible a longer piece of wire lying anywhere in the room should work. Admittedly indoor antennas suffer greater interference levels and advice on how to make optimized antennas will be printed soon in Elektor.

Construction

The PCB (**Figure 5**) is designed as an Arduino Shield, enabling it to be plugged straight into an Arduino Uno. As the SI5351 is available only as a miniature 10-pin SMD package, we took the decision to design the complete circuit with SMDs and to offer the ready-assembled PCB in the Elektor Shop [3]. Beyond this your only other action is to solder the four Arduino-compatible female headers onto the PCB. Anyone who would prefer to build the PCB completely unassisted can download the PCB layout at [3] or you can buy the PCB on its own from the Elektor Shop.

Setting the frequency

The Arduino, in conjunction with the SDR Shield, serves as an interface between antenna and PC. Its sole task is to tune the VFO and to do this, a PC program tells it which frequency is currently desired. Data from the PC reaches the Arduino via the USB cable. The downconverted signal of interest is then sent for further processing along a stereo cable to the sound card input. You could certainly attempt to relocate the control function to the Arduino as well, perhaps even some simple signal processing but this would be hard labor for a small system. For the moment it is sufficient for the Arduino to receive commands from the PC and adjust the VFO.

How you deal with the Arduino is not a topic for this article. Using the Arduino IDE is presupposed. First of all a suitable Arduino program is loaded. Exactly what happens in the software will be explained presently. However, if you're in a rush for practical results, you can skip this information and simply load the software [3].

The critical task lies in persuading the SI5351 to generate an appropriate frequency. The IC has two internal PLLs and three outputs (see block diagram, **Figure 6**). Here only PLL A and the output CLK1 are used. The concept makes use of the Adafruit Library, which makes the whole business delightfully simple. Before you let loose, you must download the Library from [2] and integrate it.

The SI5351 has a 25 MHz crystal oscillator and two PLLs that can be set between 600 MHz and 900 MHz. The PLL dividers operate with fractional division ratios, so it is in fact possible to achieve almost any desired resolution. The following multisynth divider also uses fractional division ratios. This gives you two ways to generate the desired frequency:

- You can set the PLL on a fixed frequency e.g. 900MHz and then divide

Advertisement

HAMMOND MANUFACTURING®

Housings for Raspberry Pi, Arduino and many other bareboard computers
- Enclosure
- Platform

+ 44 1256 812812
sales@hammondmfg.eu

The choice is yours.
- Enclosure for all round protection
- Platform for all round access
- Design-specific versions for all popular models
- Visit hammondmfg.com for full details

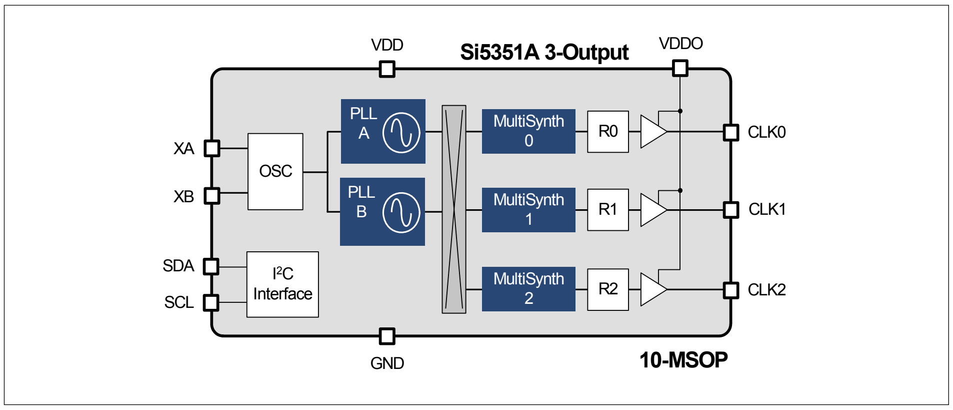
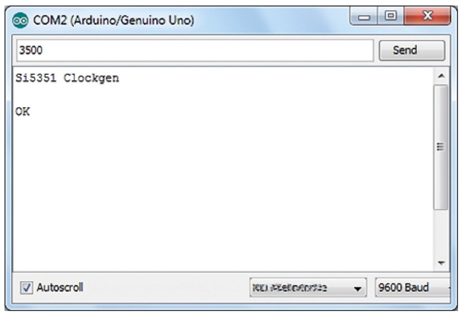


Figure 6. Block diagram from the data sheet for the SI5351A.



COM2 (Arduino/Genuino Uno)

```
3500
```

Send

```
Si5351 Clockgen

OK
```

☑ Autoscroll

9600 Baud

Figure 7. Activating the clock generator with the Arduino terminal.

down with fractional numbers.

- You can adjust the PLL in small steps and then divide by integers to reach the final frequency.

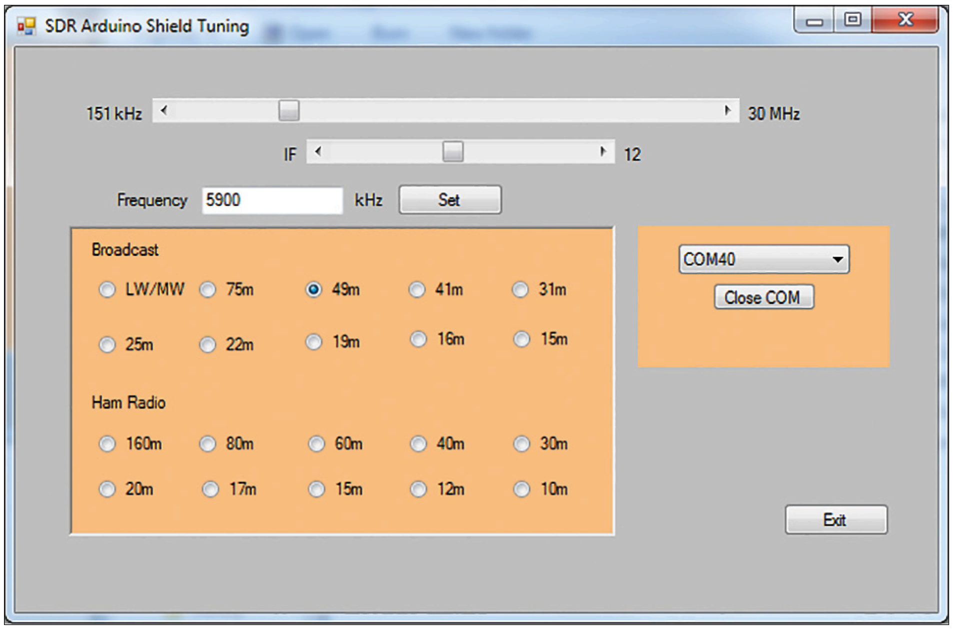
First, here is method A. The VFO frequency is four times the mixer frequency, which is 12 kHz below the receive frequency. The program is arranged to receive the radio frequency in kHz and implement in text format. In order to receive 3500 kHz the SI5351 must produce at output 1 of the SI5351 must produce an output frequency of 4 × (3500-12) = 13952 kHz. The PLL divider is set to 36 (25 MHz × 36 = 900 MHz) and the multisynth divider 900000/13952 = 64.506. Using this method we can get down to 1 MHz. For even smaller frequencies the additional R_DIV divider is set to 16. **Listing 1** indicates the relevant

Listing 1. Program for fixed-tuned PLL.

```
//SI5351_vfo  PLL fixed at 900 MHz (si5351vfo2.zip)

#include <Adafruit_Sensor.h>
#include <Wire.h>
#include <Adafruit_SI5351.h>

Adafruit_SI5351 clockgen = Adafruit_SI5351();
void setup(void)

{
  Serial.begin(9600);
  Serial.println("Si5351 Clockgen"); Serial.println("");


  /* Initialise the sensor */
  if (clockgen.begin() != ERROR_NONE)
  {
    Serial.print("Error");
    while(1);
  }
  Serial.println("OK");
  clockgen.enableOutputs(true);
  clockgen.setupPLL(SI5351_PLL_A, 36, 0, 1000);  //900
MHz
  setfreq (6000);
}

void setfreq (unsigned long freq)
{
  unsigned long f2;
  unsigned long f3;
  unsigned long f4;
  unsigned long f5;
  unsigned long div2;
  unsigned int Divider2;
  unsigned int rdiv;

   if (freq > 0)    {
    f2=(freq-12)*4;
    if (f2<1000) {
      rdiv = 16;
      f2 = f2 * 16;
      }
    else  {
      rdiv = 1;
      }
    div2 = 900000000/f2;
    f4 = div2/1000;
    f5=div2-(f4*1000);
    clockgen.setupMultisynth(1, SI5351_PLL_A, f4, f5,
1000);
    if (rdiv == 16) {
      clockgen.setupRdiv(1, SI5351_R_DIV_16);
      }
    if (rdiv == 1) {
      clockgen.setupRdiv(1, SI5351_R_DIV_1);
      }
   }
}

void loop(void)
{
  unsigned long freq;
  if (Serial.available()) {
   freq = Serial.parseInt();
   setfreq (freq);
   }
}
```

software for the Arduino; **Figure 7** shows the control panel on the Arduino terminal. Method A has the advantage that the VFO can be adjusted more or less continuously, i.e. there is no interruption when a change of frequency occurs.
On the other method B promises greater phase accuracy, adequate even for DRM. Against this, every frequency change is accompanied by a brief interruption of about a millisecond, which appears an interference signal on the SDR. The method requires calculation of the optimal fractional division (**Listing 2**) in order to keep the PLL constantly in the range 600 MHz to 900 MHz.

Both programs can be controlled by any Terminal program of your choice. However, for really convenient operation a VB program was written in Visual Studio 2015 (SDRShield.zip, downloadable at [3]). This sends the desired frequency to the Arduino in text format (e.g. 3500) at 9600 Baud. The slider control (see **Figure 8**) operates in 9 kHz steps in a range up to 1.6 MHz and beyond that with 5 kHz resolution. Additionally you can enter a desired frequency direct or click the 'band' buttons at the beginning of the individual broadcast or amateur radio bands. The first time you do this, take care that you have selected the correct COM Port.



Frequency 5900 kHz

[✓ 49m]

COM40

Figure 8. A short VB program ensures ease of operation.

SDR software

Here is a survey of the SDR software used. Practically all the programs used with the old Elektor SDR still work fine.

- *SDRadio* is still a good choice;
- *SoDoRa* can also decode DRM;
- *DREAM* still works but does not make use of the IQ signal and uses the receiver like a direct mixer;
- *HDSDR* is a current and very powerful program;
- *SDRSharp (SDR#)* is distinguished by simple operation and good on-screen representation.

In a follow-up article we shall discuss these individual programs and their possibilities in detail.

First results with reception

If you have no better antenna to hand, for your first tests you can simply connect a one-to-three meter length of wire to the antenna input. This will enable you to receive all broadcast stations on all shortwave bands without any problem. Experience shows that there is more happening at night than during the daytime. And during the evening we find the main focus is on the lower bands, between 75 m and 41 m. Even amateur radio stations can be pulled in with just a short wire aerial. Normally you'll have the best luck in the 40 meter band, where you can hear some CW and SSB stations. You can select the appropriate operational mode in the SDR software, plus audio volume, bandwidth, ALC settings and much more. With the correct settings you can often get better results than with an expensive analog receiver of the older kind.
A fundamental characteristic of all switching mixers is that signals on uneven (odd) multiples of the base frequency can also

SDR software

The current stars in the SDR software firmament are SDR# [4] and HDSDR [5]. Both programs follow the new trend for ever higher frequencies and can be driven using simple DVB-T [6] dongles. This is a good choice if you wish to poke around on the VHF and UHF bands. There have also been attempts to use this kind of hardware below 30 MHz. You can, for example, use an up-mixer that shifts every frequency 50 MHz higher. You do of course then have a multiple conversion superhet along with its well-known problems, such as countless phantom and spurious signals together with reduced dynamic range. A dedicated SDR for frequencies up to 30 MHz uses single conversion only and in that way manages to deliver very clean reception without 'birdy' whistles.

On your PC you have two programs running, namely the tuning program and the SDR software. Each SDR program has its own method of operation but the basic steps are nevertheless similar in each case. First you need to establish that the correct input is in use. For this you need to select the sound card and activate the chosen input (Line In). Next you boot up the SDR software. You'll know you have selected the correct input when you see a significant rise in the noise floor, which should increase still more after connecting the antenna. Most sound cards need to have their volume control throttled back, as the receiver can deliver up to a volt of output signal.

Listing 2. Program for variable PLL.

```
//SI5351_vfo, variable PLL (si5351vfo3.zip)

#include <Adafruit_Sensor.h>

#include <Wire.h>
#include <Adafruit_SI5351.h>

Adafruit_SI5351 clockgen = Adafruit_SI5351();
void setup(void)

{
  Serial.begin(9600);
  Serial.println("Si5351 VFO"); Serial.println("");

  if (clockgen.begin() != ERROR_NONE)
  {
    Serial.print("Error");
    while(1);
  }
  Serial.println("OK");
  clockgen.enableOutputs(true);
  setfreq (6000);
}

void setfreq (unsigned long freq)
{
  unsigned long f2;
  unsigned long f3;
  unsigned long f4;
  unsigned long f5;
  unsigned int Divider2;
  unsigned int rdiv;

   if (freq > 0)
   {
   f2=(freq-12)*4;
   // f2=freq;
    if (f2>120000) {
      f2=120000;
    }
    if (f2<800) {
       rdiv = 16;
       f2 = f2 * 16;
    }
    else  {
       clockgen.setupRdiv(1, SI5351_R_DIV_1);
       rdiv = 1;
    }
    if (f2 >= 100000) {
      Divider2 = 6;
     }
     if (f2 < 90000) {
      Divider2 = 10;
     }
     if (f2 < 60000) {
      Divider2 = 15;
     }
     if (f2 < 50000) {
      Divider2 = 18;
     }
     if (f2 < 45000) {
      Divider2 = 20;
     }
     if (f2 < 30000) {
      Divider2 = 30;
     }
     if (f2 < 20000) {
      Divider2 = 45;
     }
     if (f2 < 15000) {
      Divider2 = 60;
     }
     if (f2 < 10000) {
      Divider2 = 90;
     }
     if (f2 < 6000) {
      Divider2 = 150;
     }
    if (f2 < 4000) {
      Divider2 = 220;
     }
    if (f2 < 2700) {
      Divider2 = 330;
     }
    if (f2 < 1800) {
      Divider2 = 500;
     }
    if (f2 < 1500) {
      Divider2 = 600;
     }
    if (f2 < 1000) {
      Divider2 = 900;
     }
    f2=f2*Divider2;
    f2=f2*1000/25;
    f3=f2 /1000;
    f4 = f3/1000;
    f5=f3-(f4*1000);
    clockgen.setupPLL(SI5351_PLL_A, f4, f5, 1000);
    clockgen.setupMultisynth(1, SI5351_PLL_A, Divider2,
      0, 2);
    if (rdiv == 16) {
      clockgen.setupRdiv(1, SI5351_R_DIV_16);
    }
  }
}

void loop(void)
{
  unsigned long freq;
  if (Serial.available()) {
   freq = Serial.parseInt();
   setfreq (freq);
  }
}
```

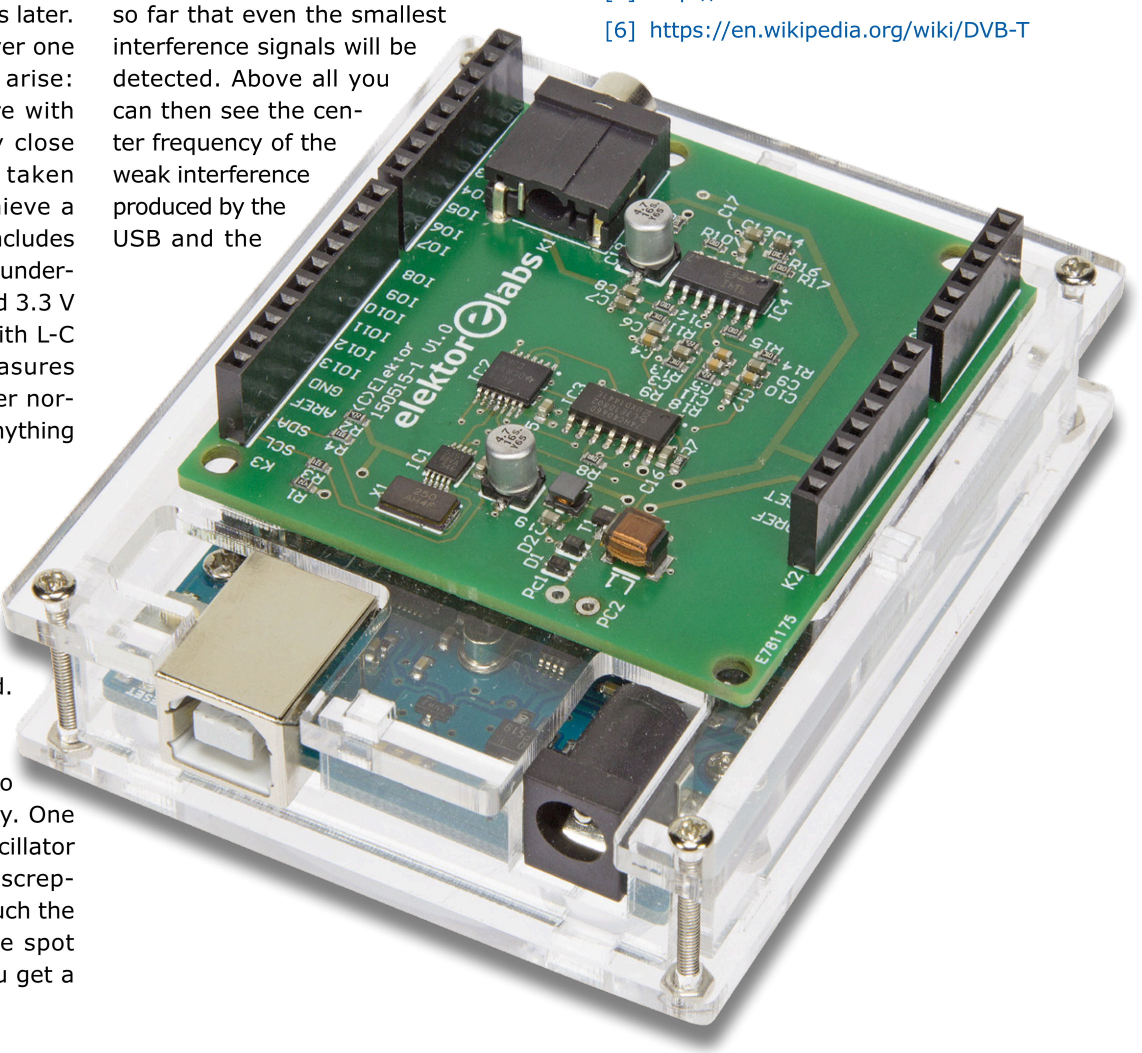
be downconverted. If you want to receive a signal on 1 MHz, other signals on 3 MHz, 5 MHz, 7 MHz and so on can disturb your reception. For this reason people often use switchable lowpass filters. The SDR Shield doesn't include one of these, so it makes sense to use an antenna that's selective. Even so, things work astonishingly well with a wideband wire antenna. The reason for this is that at specific times of the day strong signals dominate on various bands and get through unscathed. An exception to this is reception on long and medium wave, which can be desensitized by signals in the short wave region. You can eliminate this problem by using a ferrite rod antenna with a rotary tuning capacitor.
The theme of antennas, filters and preamplifiers needs to be examined in closer detail. This involves not merely large signal voltages but also the achievable signal-to-noise ratio. There's nothing better than a long wire antenna, erected as far as possible from your house for this. But because this is not always possible, we must look for compromises. And in this respect the magnetic loop antenna is the clear winner. These enable you to have yourself a relatively small and unobtrusive antenna indoors. More on this later.
In your first trials with this receiver one particular question is bound to arise: won't the Arduino itself interfere with reception? It is after all in very close range. In fact great care was taken when laying out the PCB to achieve a high degree of decoupling. This includes a continuous ground plane on the underside of the PCB, whilst the 5 V and 3.3 V supply voltages are decoupled with L-C filters. In actual fact these measures are extremely effective and under normal conditions you won't notice anything from the Arduino.

Eavesdropping with the Arduino

What you might at least 'receive', however, is the 16 MHz clock oscillator. This will occur when you have no antenna at all connected. The Shield can then demonstrate its ability to function as a test device. Actually there are two oscillators running simultaneously. One of these is the 16 MHz crystal oscillator on the Uno's USB chip with a discrepancy of less than 1 kHz. If you touch the underside of the Uno PCB at the spot where the crystal is soldered, you get a slight amount of detuning. You know then that it was the signal in question. With a short piece of wire on the antenna input the signal will grow stronger, as will the noise floor. And we can exploit this fact: signals that arrive via the antenna input have good image frequency suppression, whereas it's different for those that creep into the signal path via the supply voltage. The latter exhibit twice the frequency but are significantly weaker.

The clock signal of the Mega328 is another thing to track down. This oscillator uses a ceramic resonator and can exhibit discrepancies of up to 50 kHz. In point of fact a weak signal was found on 15950 kHz with some sideband signals into the bargain, contributed by the controller. Touching the Arduino PCB in the region of the ceramic resonator additionally set off some broad FM modulation and further detuning, which proved that the resonator was temperature-dependent to some degree. Of course it's only when you have an SDR that you can sound out the Arduino so accurately!
Without an antenna connected a SDR will normally crank up the amplification so far that even the smallest interference signals will be detected. Above all you can then see the center frequency of the weak interference produced by the USB and the Arduino. If you wanted to separate out the signals caused by the Arduino from those coming from the USB, you could simply power up the Arduino and after tuning into your desired frequency, unplug the USB cable leaving the rest running.
All internal interference signals are very weak though. As soon as you connect an antenna, the noise floor rises to the extent that all the interference is entirely masked. This shows the high sensitivity of the SDR. Even signals of only one microvolt can be received. Normally this level of sensitivity is entirely unnecessary, because the noise level of the antenna is significantly higher. Using long antennas can even lead to overmodulating the receiver and in these situations you will have to consider using an input attenuator. ◀

(150515)



Web Links

[1] www.elektormagazine.com/070039
[2] https://github.com/adafruit/Adafruit_SI5351_Library
[3] www.elektormagazine.com/150515
[4] http://airspy.com/download
[5] http://www.hdsdr.de
[6] https://en.wikipedia.org/wiki/DVB-T