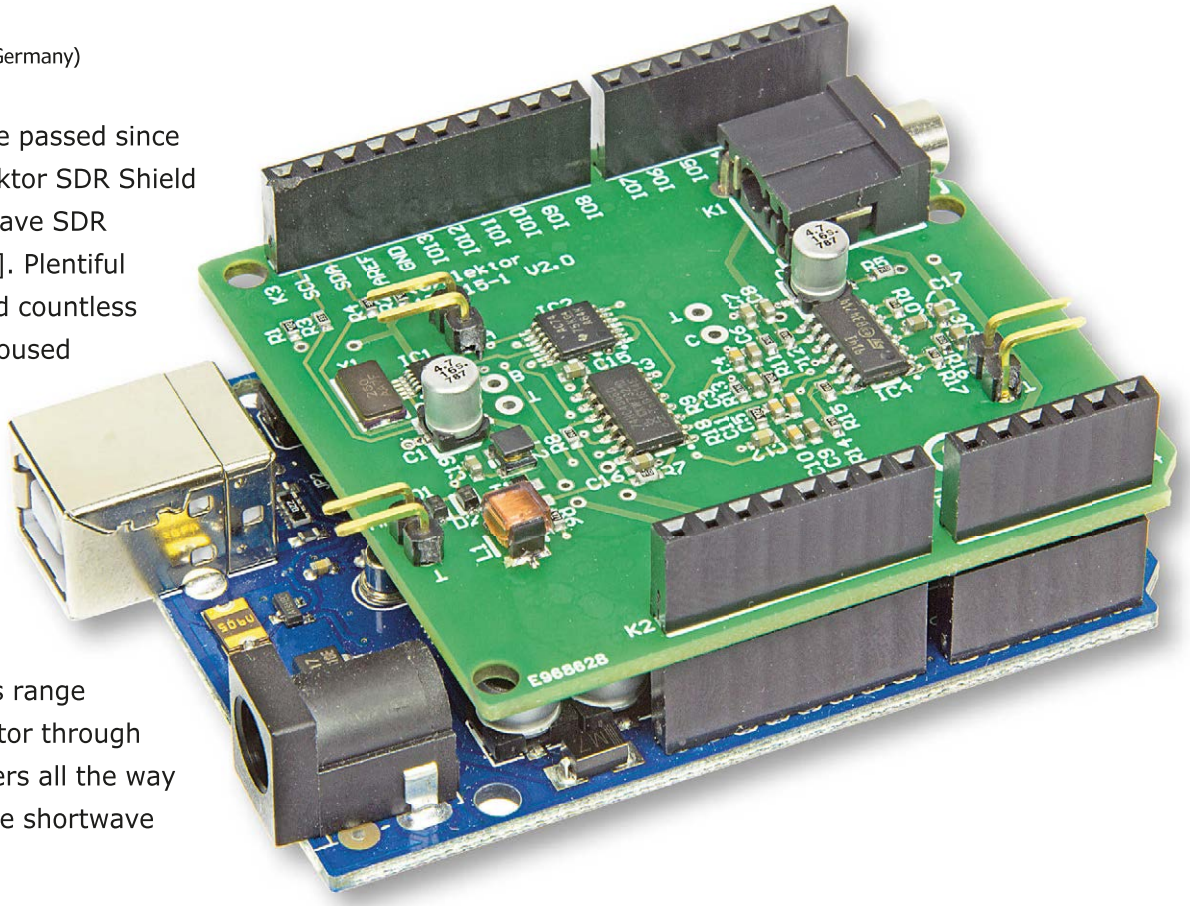
Elektor SDR Shield 2.0 (1)

Reception and tuning with this versatile new device

By **Burkhard Kainka** (Germany)



About two years have passed since the launch of the Elektor SDR Shield for creating a shortwave SDR using the Arduino [1]. Plentiful positive feedback and countless experiments have aroused the desire to provide greater connectivity onboard. SDR Shield 2.0 is now the basis for far more than simply a receiver. Applications range from a signal generator through shortwave transmitters all the way to forming a complete shortwave transceiver.

A feature of the first version of the SDR Shield was the SI5351, a triple PLL generator of which only one output was used with the receiver. It was soon apparent that users could tackle plenty more if provided with at least one more output. With this enhancement the SDR Shield could then be expanded into a complete CW transceiver for amateur radio use. Up till now, however, this output was not easily accessible on the SMD module. For the second version of the board we have provided direct connections to two additional outputs from the PLL generator. You simply solder a couple of pinheaders and attach the necessary cables.

There's also much more you can do besides amateur radio. The Shield can now become a universal signal generator with frequency accuracy that can be trimmed to the highest standards using software alone. Every electronics lab needs something like this. Or by combining the sig gen and receiver functions you can create a level measurement instrument for analysing frequency response and impedance curves. Everybody can now devise their own particular measurement tasks with a simple Arduino Sketch. The schematic (**Figure 1**) is barely altered since the previous version. Only the four connections A through D are newly added.

A: SI5351 output CLK0
B: SI5351 output CLK2
C: AF/IF left output, DC-coupled
D: AF/IF right output, DC-coupled

Figure 2 illustrates the Shield and the 90-degree pinheaders for connecting the

Characteristics

- Operating voltage: 5 V and 3.3 V from Arduino
- Frequency range: 150 kHz to 30 MHz
- Sensitivity: 1 µV
- Overall gain: 40 dB
- Maximum signal level at antenna: 10 mV
- Dynamic range: 80 dB

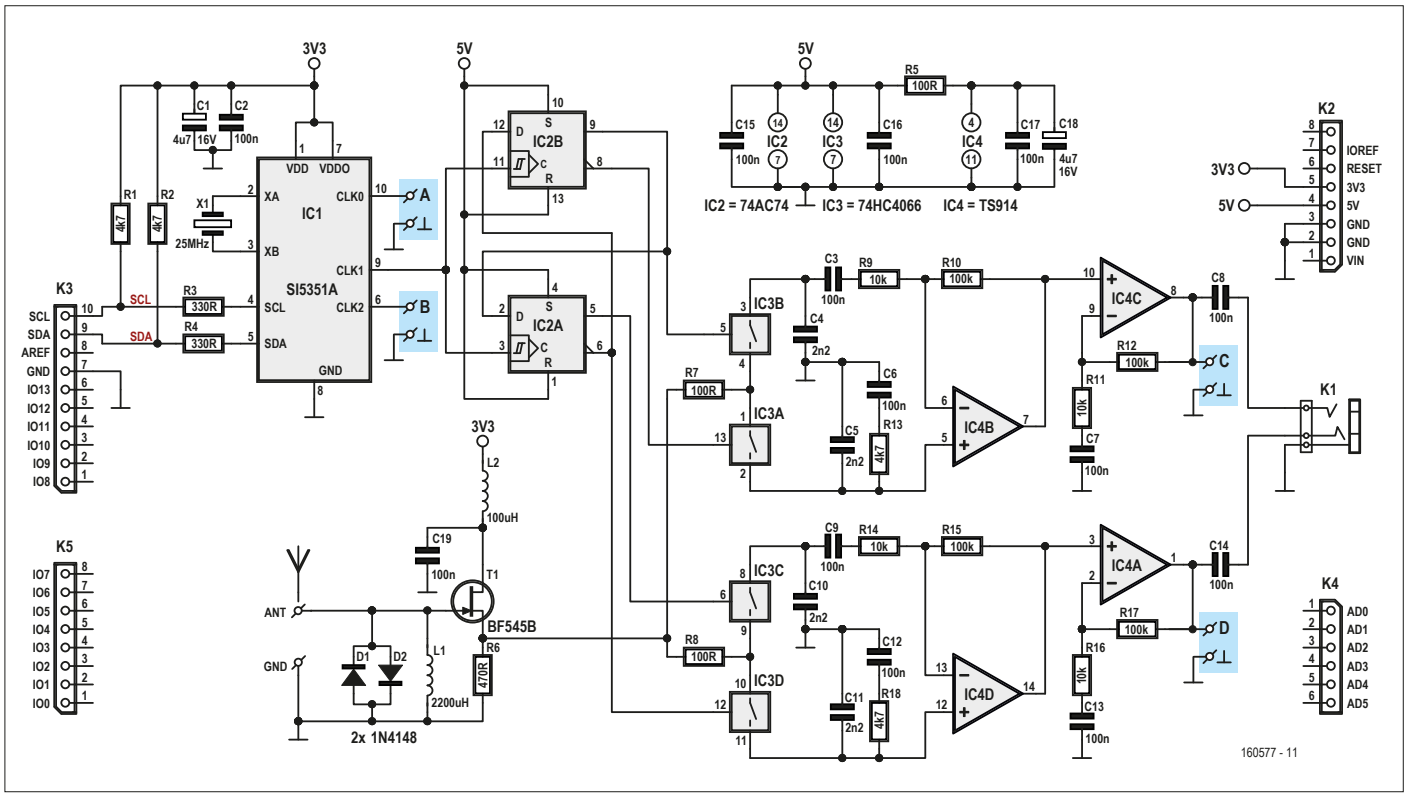


Figure 1. The new connections are highlighted in the schematic.

antenna and two auxiliary connections. They are placed in such a way as to remain compatible with the LCD-equipped Elektor Experimenting Shield [2]. This approach was already validated in the first version. In this way autonomous (stand-alone) receivers can be created even without the need for a supporting PC. With version 2.0 individual standalone measurement devices can also be created.

First steps with G8JCFSDR

As not all readers will be familiar with the original SDR Shield, we ought to describe its beginnings here briefly. The simplest entry point is using the G8JCFSDR SDR program (**Figure 3**) by Peter Carnegie [3]. It was introduced in Elektor Magazine

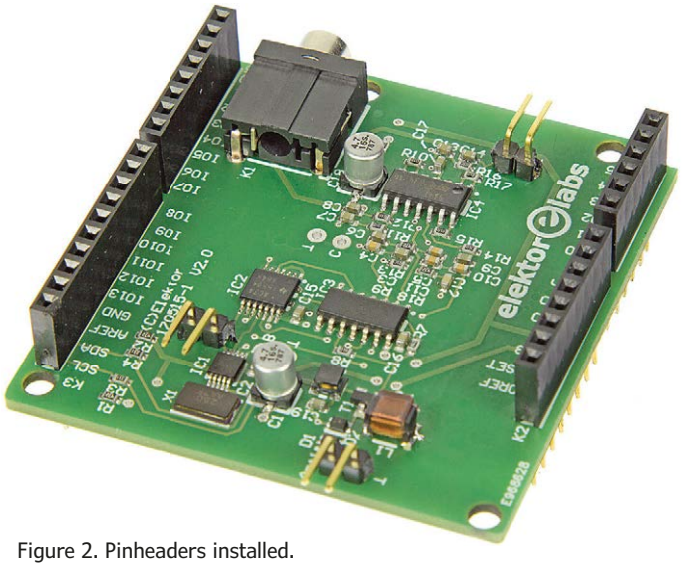


Figure 2. Pinheaders installed.

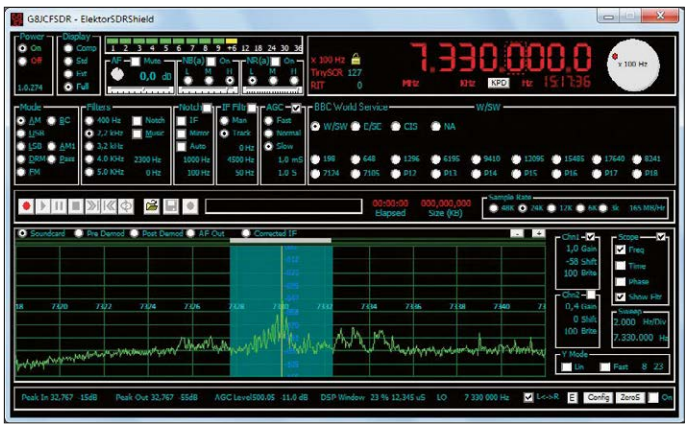


Figure 3. Receiving a radio station.

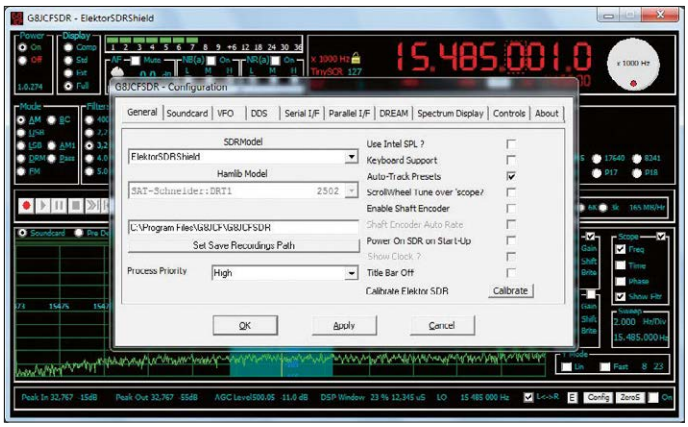


Figure 4. Selecting the receiver hardware in use.

back in 2007 [4] and has been expanded continuously since then. Tuning has now become feasible down to an accuracy of a single Hz and individual calibration of the VFO is now child's play. The program loads the appropriate firmware into the Arduino entirely automatically. So you don't even need to involve the Arduino IDE. Simply connect up, switch on and you're ready for action.

At initial switch-on the SDR is still 'off'. Your first task is to select the hardware used (**Figure 4**), then also define the interface employed (e.g. COM2) under the heading *Serial I/F*. At first activation the program determines whether the necessary firmware is present. If not, a window then opens for an automatic upload (**Figure 5**). This is a great help for developers who have otherwise not had much experience with the Arduino. No more problems with compilers, software versions and other tribulations of this kind.

The basic accuracy of the VFO will not suffice for many applications because the 25-MHz oscillator can exhibit discrepancies of several kHz. Calibration assists here. You tune to a radio station on a known frequency, click then on *Calibrate* and receive a further menu (**Figure 6**). Now you tune in the transmitter as accurately as you can and click on *Apply*. The VFO is now calibrated. Incidentally, for absolutely optimal calibration you should, in the *USB* setting,

COMPONENT LIST

Resistors

R1,R2,R13,R18 = 4.7kΩ, 1% 100mW, SMD 0603
R3,R4 = 330Ω, 1% 100mW, SMD 0603
R5,R7,R8 = 100Ω, 1% 100mW, SMD 0603
R6 = 470Ω, 1% 100mW, SMD 0603
R9,R11,R14,R16 = 10kΩ, 1% 100mW, SMD 0603
R10,R12,R15,R17 = 100kΩ, 100mW, 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 = 2.2µF 50V, X7R, SMD 0603

Inductors

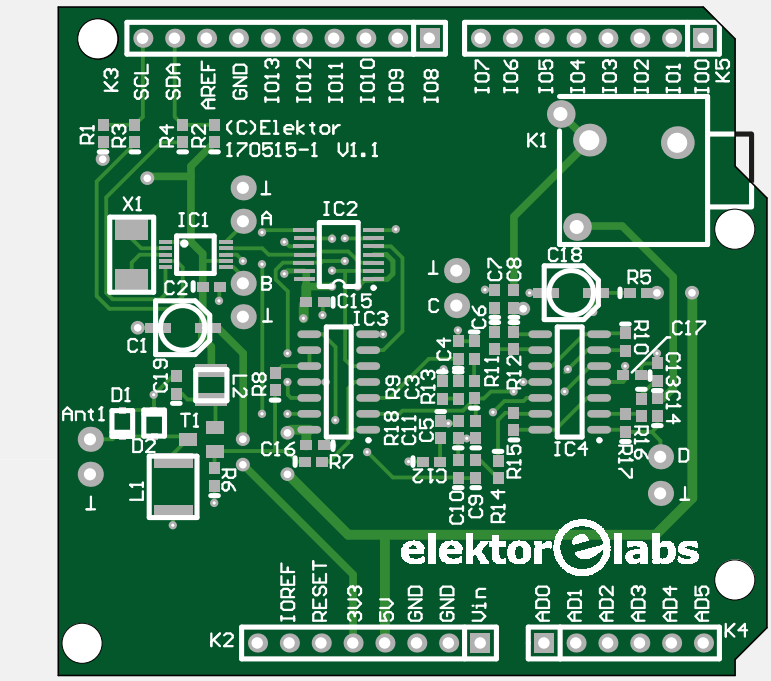
L1 = 2200µH (Fastron L-1812AF)
L2 = 10µH (Murata LQH32CN101K23L)

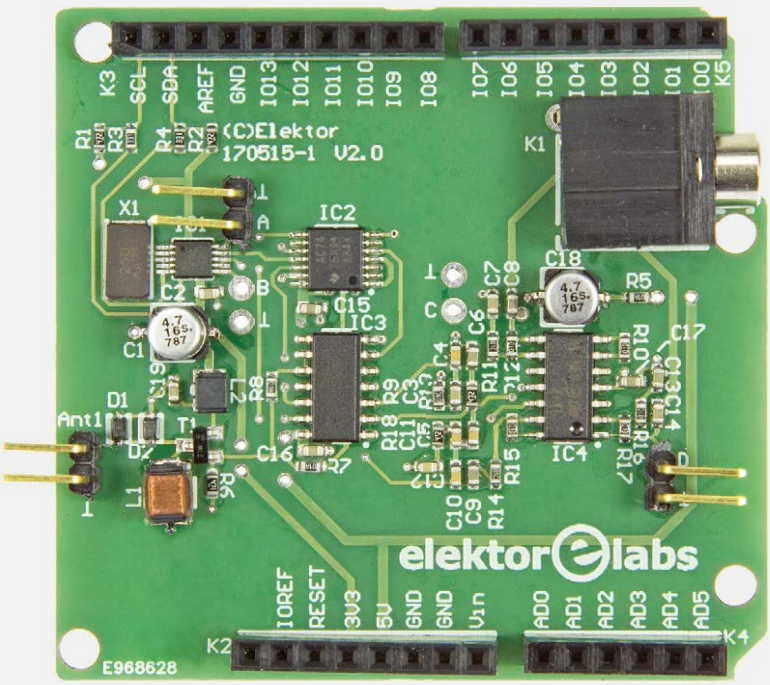
Semiconductors

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, 3.5mm, PCB mount
K2,K3,K4,K5 = 1 set of Arduino-compatible interconnection socket strips, (1x 6-pin, 2x 8-pin, 1x 10-pin)
X1 = 25-MHz quartz crystal (Abracon ABM7)
PCB 170515-1
or
PCB with SMDs preassembled: 170515-91





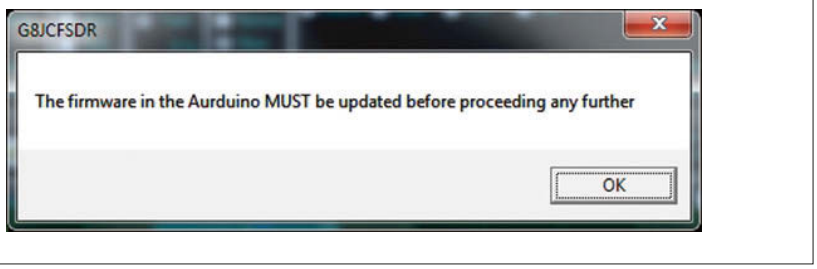


Figure 5. Confirmation of the upload.

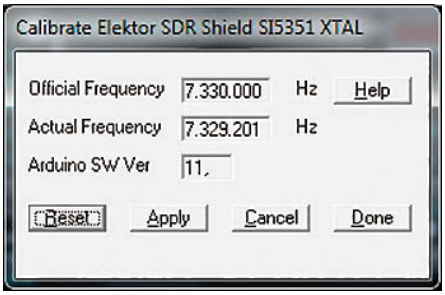


Figure 6. Calibrating the VFO.

tune in a broadcast transmitter for zero beat. The oscilloscope representation of the audio frequency outputs can assist here (AF OUT, Scope -> Time). According to how the Arduino was used previously, an entirely false calibration may arise on the first run. If so, a click on *RESET* is enough to enforce the default settings, in which an error of a few kHz can be expected.
With this the receiver is now ready for action. It enables you to monitor all signals from AM broadcast radio, CW and SSB. Digital operational modes can also be decoded if you provide suitable extra software.

Antenna advice

When you first go forth on the airwaves using SDR you may experience a nasty surprise, simply because there is little to be heard other than loud hissing. For the benefit of newcomer users here are some tips about antennas suitable for SDR. For powerful broadcast transmitters on the short waves a wire one metre in length, hanging or lying anywhere in the room, is entirely satisfactory. Unfortunately this indoor antenna will also pick up interference from power wiring and electrical appliances. Comprehensive digitalisation has led to powerful background noise that makes your quest more challenging, particularly indoors.

You can improve the aerial by increasing the length of the antenna wire and erecting it outdoors. Unfortunately even more interference then reaches the receiver input, arising mainly from the contaminated GND (earth wire) connection. Even if the antenna delivers a totally clean signal, its counterpoise is the contaminated earth connection. Both voltages are summed at the input, with the nett result that you suffer an elevated noise floor that drowns any weak signals.

A well-respected remedy can achieve a less contaminated earth connection. If a direct ground connection is not an option you can employ a cold water or central heating pipe as a substitute. In addition, you will need an RF transformer-type isolating barrier/coupler that prevents the clean earth connection from being polluted by the protective ground wire. In the ideal scenario a coaxial cable leads from the receiver to the transformer (**Figure 7**) placed close to the window, so that the actual antenna is located more or less entirely external to the house. With a little luck you can improve the noise threshold by up to 20 dB.

For the transformer widely differing cores are suitable. **Figure 8** shows an iron powder core with two separate windings. The size of the core and the number of turns are not critical and you can also experiment with the turns ratio.

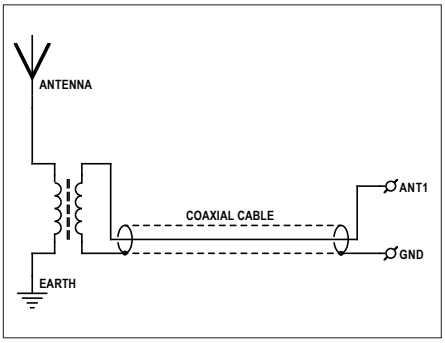


Figure 7. Installing a transformer.

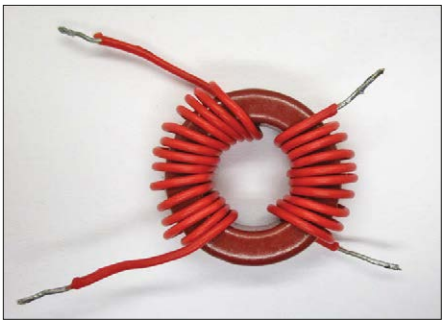


Figure 8. Isolated windings on a ring core.

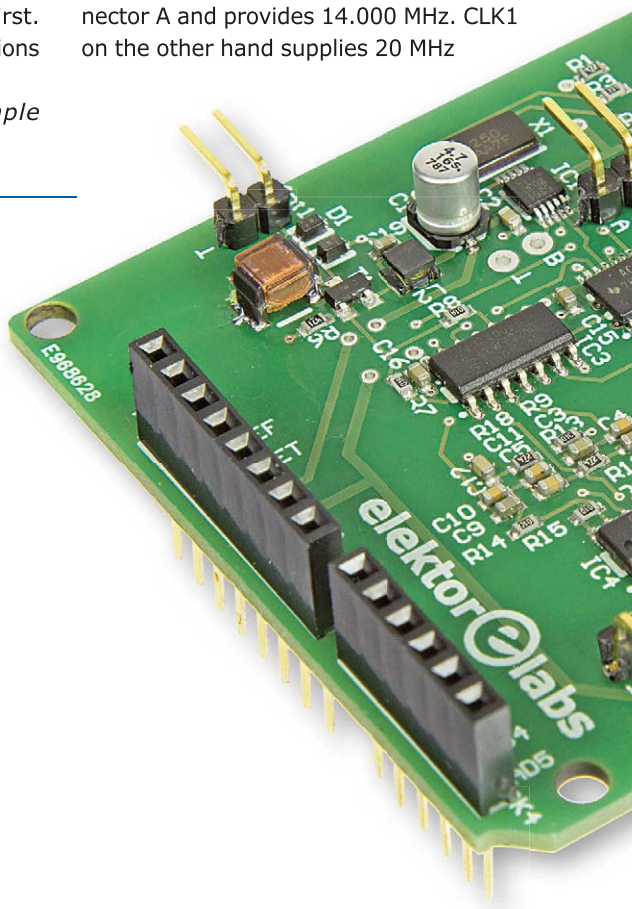
Listing 1. PLL control code in si5351example (excerpts).

```
#include "si5351.h"
#include "Wire.h"

Si5351 si5351;

void setup()
{
  // Start serial and initialize the Si5351
  Serial.begin(57600);
  si5351.init(SI5351_CRYSTAL_LOAD_8PF, 0);

  // Set CLK0 to output 14 MHz with a fixed PLL frequency
  si5351.set_pll(SI5351_PLL_FIXED, SI5351_PLLA);
  si5351.set_freq(1400000000ULL, SI5351_PLL_FIXED, SI5351_CLK0);

  // Set CLK1 to output 20 MHz
  si5351.set_freq(2000000000ULL, 0ULL, SI5351_CLK1);
}
```

VFO control using the Arduino

Now it's time to develop your own Arduino software. Nowadays the Etherkit/Si5351 Arduino Library has proved itself as the best choice when you need to control the PLL chip flexibly. This allows individual channels of the PLL module to be enabled and disabled, also offering the option of setting the phase between two channels with the same frequency. Accurate frequency calibration is another possibility. It is worth checking out the examples included with the library first. The library and some sample applications are available at [5].

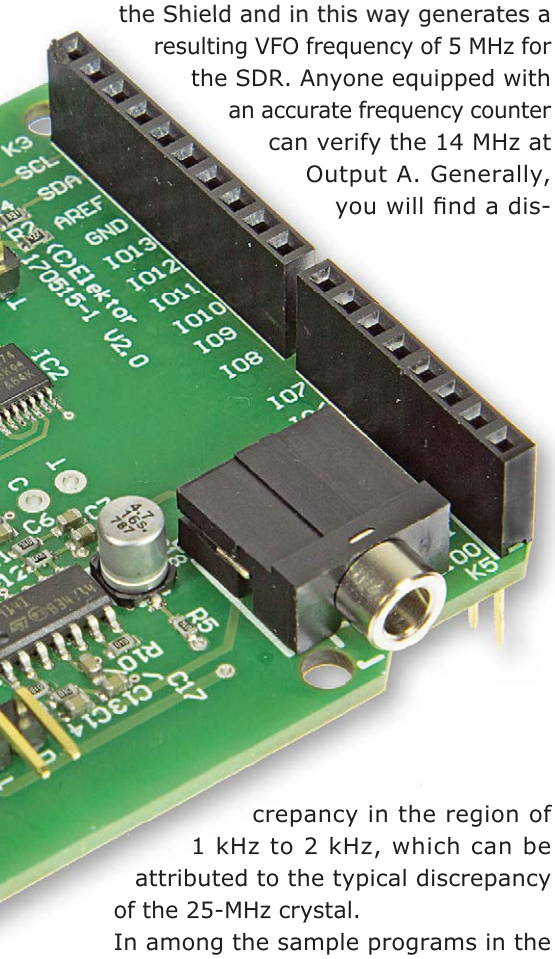
The sample code in *si5351example* (**Listing 1**) shows the principles of how this works. It can be used with the SDR Shield straightaway without modification. Output CLK0 corresponds to VFO connector A and provides 14.000 MHz. CLK1 on the other hand supplies 20 MHz



Listing 2. Implementing control commands in *s15351vfo2_0*.

```
void loop(void)
{
  si5351.update_status();
  if (si5351.dev_status.SYS_INIT == 1) {
     setup();
     delay(500);
  }
  if (Serial.available()) {
    int ch = Serial.read();
    freq = Serial.parseInt();
    Serial.println(ch);
    Serial.println (freq);
    int ch2 = Serial.read();
    Serial.println (ch2);
    lcd.setCursor(0, 1);
    lcd.print(ch);
    lcd.print(" ");
    lcd.print(ch2);
    ch2 = Serial.read();

    if (freq> 20){
      lcd.setCursor(0, 0);
      lcd.print(freq);
      lcd.print("     ");
      if (ch == 32) si5351.set_freq(freq*400000ULL, SI5351_PLL_FIXED, SI5351_CLK1);  //" "
      if (ch == 70) si5351.set_freq(freq*400000ULL, SI5351_PLL_FIXED, SI5351_CLK1);  //F
      if (ch == 102) si5351.set_freq(freq*400ULL, SI5351_PLL_FIXED, SI5351_CLK1);    //f

      if (ch == 65) si5351.set_freq(freq*100000ULL, SI5351_PLL_FIXED, SI5351_CLK0);  //A
      if (ch == 97) si5351.set_freq(freq*100ULL, SI5351_PLL_FIXED, SI5351_CLK0);     //a
      if (ch == 66) si5351.set_freq(freq*100000ULL, SI5351_PLL_FIXED, SI5351_CLK2);  //B
      if (ch == 98) si5351.set_freq(freq*100ULL, SI5351_PLL_FIXED, SI5351_CLK2);     //b
    }
    if (freq==0){
      if (ch == 65) si5351.output_enable(SI5351_CLK0, 0);
      if (ch == 66) si5351.output_enable(SI5351_CLK2, 0);
      if (ch == 70) si5351.output_enable(SI5351_CLK1, 0);
    }
     if (freq==1){
      if (ch == 65) si5351.output_enable(SI5351_CLK0, 1);
      if (ch == 66) si5351.output_enable(SI5351_CLK2, 1);
      if (ch == 70) si5351.output_enable(SI5351_CLK1, 1);
    }
  }
}
```

at CLK1. The signal is divided by 4 on the Shield and in this way generates a resulting VFO frequency of 5 MHz for the SDR. Anyone equipped with an accurate frequency counter can verify the 14 MHz at Output A. Generally, you will find a discrepancy in the region of 1 kHz to 2 kHz, which can be attributed to the typical discrepancy of the 25-MHz crystal.



In among the sample programs in the library we also find *si5153calibration*, with which you can fine-tune a frequency of exactly 10 MHz. Using the Arduino Terminal program, the frequency can be calibrated in small increments. At the end of the code there is a factor that you can employ effectively in your own programs as well: `si5351.set_correction(cal_factor);`. This achieves the best accuracy with minimal discrepancies of just a few hertz.

If you still suspect that your own frequency counter is not sufficiently accurate, or don't have one to hand, there exists yet another method. You can make use of a broadcast transmitter of known frequency and generate a signal on the same frequency, whereby any discrepancy will be audible as a beat frequency. For this task you can use any shortwave receiver of your choice. Here we used Radio China on 9525 kHz and then entered this frequency in the sample program. First of all, a superimposed beat note of around one kilohertz appeared. The frequency was then adjusted for zero beat.

In the Terminal you can see the individual key commands and can now increase or decrease the frequency. In this way you can take smaller and smaller steps to achieve the correct calibration (**Figure 9**).

The end result was a calibration factor von 154400. It should be understood that the frequency with regard to 10 MHz was too high by a factor of 154400 Hz. To correct this you need to write in your program as follows: `si5351.set_correction(154400);`. You could of course determine the discrepancy of the receiver using other methods and then convert accordingly to 10 MHz.

Versatile VFO firmware

Our new VFO firmware ought to be compatible with the first version and work with previous PC programs, at the same time supporting the additional new channels (download it at [6]). But the extended requirements should be considered as well. Up till now the old firmware handled the receive frequency only in kHz at 9600 Baud and assigned this accordingly. In the new firmware, the command F still performs tuning in whole kHz but using a lower-case f now does the same thing in Hz. In addition, there are now commands for the additional channels A and B. These two outputs should be enabled only when needed, so there are short commands for switching them on and off.

F 3500	kHz tuning (VFO = 3500 kHz)
f 3500250	Hz tuning (VFO = 350250 Hz)
A 7000	Output A, kHz resolution
a 7000300	Output A, Hz resolution
A 0	Output A disable
A 1	Output A enable
B 7000	Output B, kHz resolution
b7000300	Output B, Hz resolution
B 0	Output B disable
B 1	Output B enable

Listing 2 shows the process for reception and processing the command data. The command character (F, f, A, a etc.) is placed in the Variables *ch*, the frequency in *f*. The received frequency f is also displayed on the LCD screen. An additional serial data output is provided for debugging. In this way you can observe whether the data of a PC program has been received correctly. According to the control command the frequencies are then assigned to the individual channels. You can still use the software of the previous SDR shield (**Figure 10**) to control the VFO. The only difference is that the new firmware now sets VFO frequency instead of the actual reception frequency (as long as the IF is still on 12 kHz). Clicking on the 20-metre band sets the VFO frequency to 14000 kHz. The band then starts in the middle of the spectrum. Specifying the VFO frequency has become increasingly prevalent, especially in digital modes. Incidentally, the IF slider (slide bar) control can be used for fine-tuning in 1 kHz increments.

Normally you use the tuning program in addition to the actual SDR software. **Figure 11** shows the use of SDR # (download at [7]) for receiving CW signals.

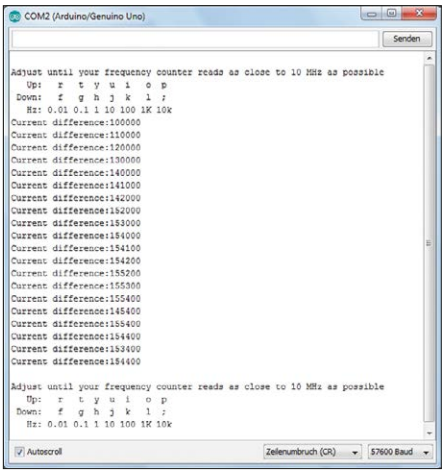


Figure 9. Calibration process using *si5153calibration* and Arduino Terminal.

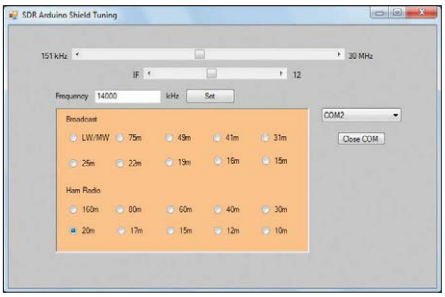


Figure 10. Tuning to 14 MHz.

A new VFO interface

The new PC program *SDRshield2_0.exe* was written in VB6 and exploits the new features of the enhanced firmware (**Figure 12**). The lower slider control has

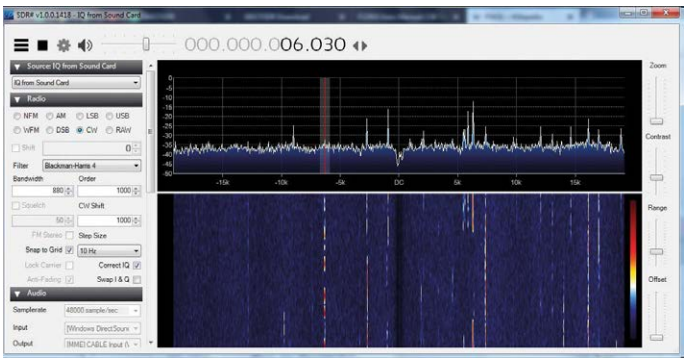


Figure 11. SDR# in use.

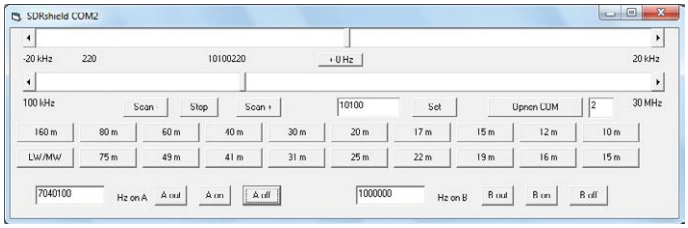


Figure 12. Setting the VFO frequency and the auxiliary outputs.

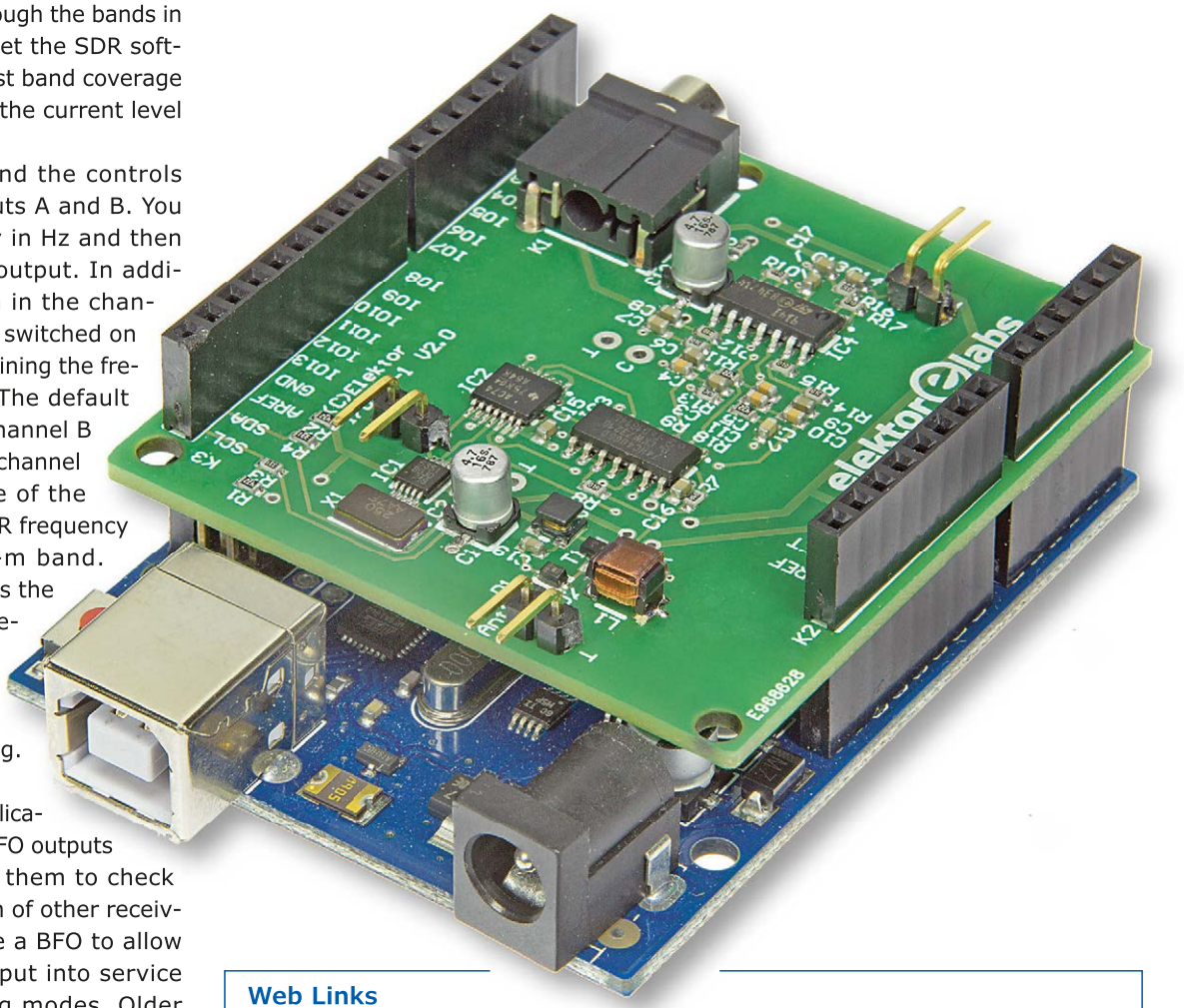
a resolution of 1 kHz and makes large steps of 25 kHz. The upper slider delivers fine tuning in increments of 10 Hz. As well as the radio buttons for individual frequency bands there is now a scanner function that works through the bands in 5-kHz steps. You can set the SDR software to AM in the widest band coverage to gain a quick idea of the current level of occupancy.

Below this you will find the controls for the auxiliary outputs A and B. You can enter a frequency in Hz and then send this to the VFO output. In addition, you must switch in the channel. The output can be switched on and off repeatedly, retaining the frequency that you set. The default is 1.000000 MHz for channel B and 7.040100 MHz for channel A, which is the centre of the only 200-Hz-wide WSPR frequency allocation on the 40-m band. This digital mode places the highest demands on frequency accuracy, for which the output on Channel A can be an aid to the correct tuning.

There are countless applications for the auxiliary VFO outputs A and B. You can use them to check the frequency precision of other receivers. Or you can create a BFO to allow AM-only radios to be put into service for SSB and operating modes. Older ham band receivers using rotary tuning capacitors are often equipped with an input for an external VFO. The SDR Shield can undertake this function and provide improved frequency accuracy in the process, which is critical with some digital operating modes.

Further tasks exist in the field of measurement engineering. With this you have two universally applicable generators covering the entire range from 8 kHz to 160 MHz. In this way you can equalise resonant circuits, verify lowpass filters or test the suitability of small ring core transformers for specific frequency ranges. Further applications in measurement technology will be the subject of another topic in Elektor Magazine. ⏮

(160577/170515)



Web Links

[1] www.elektormagazine.com/150515
[2] www.elektormagazine.com/140009
[3] www.g8jcf.dyndns.org/index.htm
[4] www.elektormagazine.com/070565
[5] https://github.com/etherkit/Si5351Arduino
[6] www.elektormagazine.com/160577
[7] https://airspy.com/download/