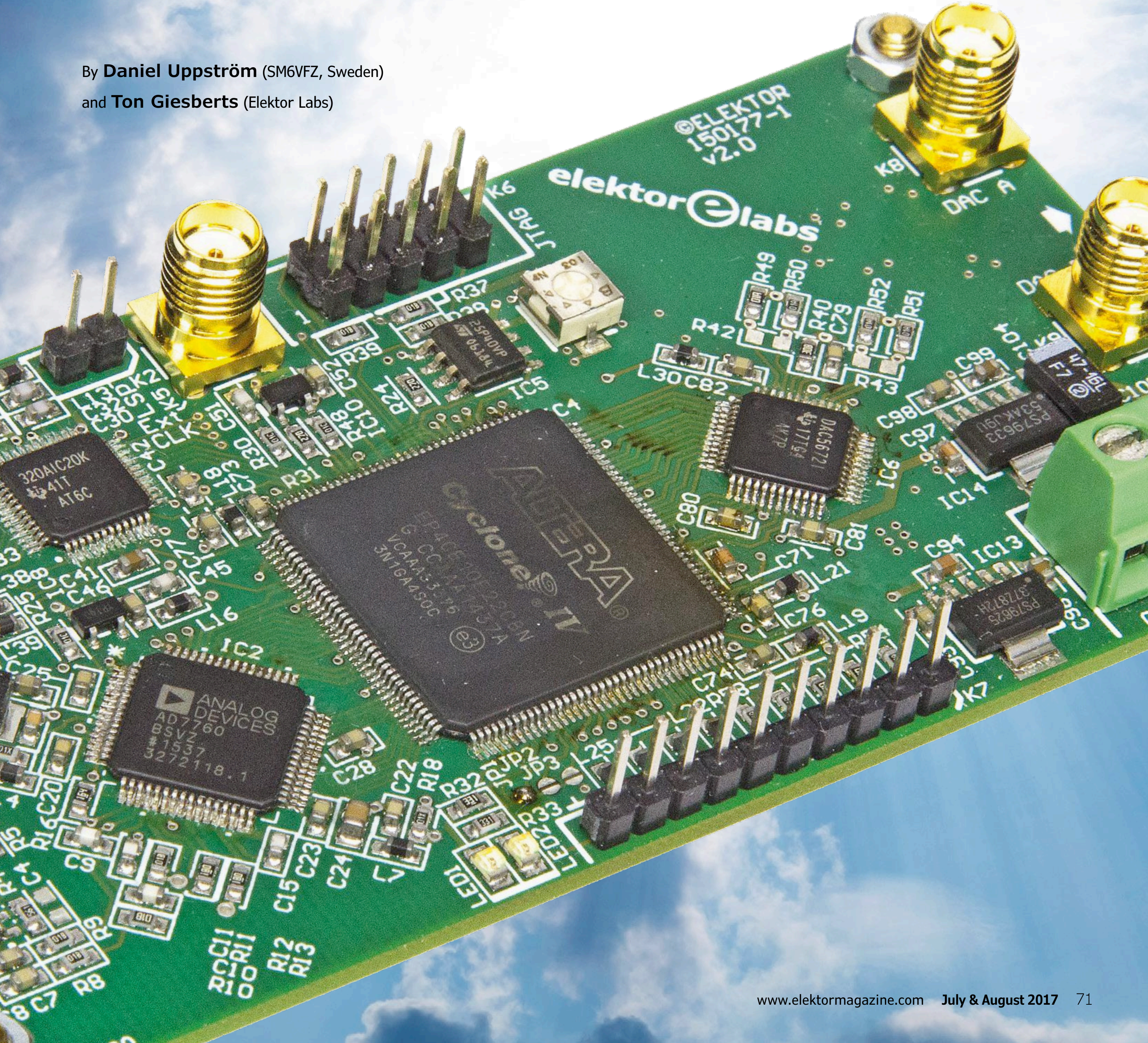
eLABS PROJECT

FPGA-DSP Board for Narrowband SDR

A fully programmable receiver/transmitter baseboard

By **Daniel Uppström** (SM6VFZ, Sweden)
and **Ton Giesberts** (Elektor Labs)



Presented here is an FPGA-based digital signal processing board featuring everything necessary to do baseband processing of traditional narrowband modes like SSB/CW and AM. It solves many of the issues and limitations seen in simpler SDR platforms. Capable of transmitting as well as receiving, it provides a perfect foundation for a shortwave and/or VHF/UHF/SHF/microwave transceiver, ideal for scientific, ham, and other serious radio applications.

From the good old days...

There once was a time when radio technology was the focus for the bulk of electronic hobbyists and electronic engineers. Many people built their own equipment and having a radio amateur license was synonymous with being interested in electronics. But when the technologies evolved it became harder for hobbyists to keep up and eventually most radio amateurs bought their equipment ready-made. The state-of-the-art transceivers of the 80's and 90's made use of a very large number of complex analog building blocks and it became almost impossible for hobbyists to build something similar, not only because of the amount of knowledge needed, but also with regard to size and costs.

With the evolution of the Internet new ways of communication arose and the number of commercially-built radio transceivers for amateurs started to decrease. At the same time many compromises were accepted in a bid to make radios smaller and cheaper while covering many frequency bands. The reduced performance with regard to essential radio properties like receiver selectivity and clean transmitted signals was to some extent compensated for by fancy digital features, color displays, and so on. However, for those who are not easily impressed by looks and gadgets and prefer a high-performance radio instead, it now again makes sense to try to build something at home.

... to software-defined radio

Still, reverting to state-of-the-art analog technology may not be the best solution and it may be more interesting to look at what is happening in the field of Software Defined Radio (SDR) where a large part of the radio signal processing is implemented in the digital domain and executed by software. With the computing power and the good analog-to-digital converters (ADCs) available today SDR technology really makes sense.

Many hobbyist and amateur radio applications process digitized signals entirely in software by a PC or a Digital Signal Processor (DSP). Although this approach is flexible, it is rather inefficient since signal processing is still relatively costly in terms of processor operations when done without hard-coded multipliers and similar peripherals. Many of the popular SDR platforms also have poor or non-existent analog filtering before digitizing, as well as low-resolution in terms of the ADC, resulting in poor performance especially with strong interfering signals around.

A better way would be to build a good analog radio front-end and do the signal processing with programmable logic. Today's Field-Programmable Gate Arrays (FPGAs) available at affordable prices sport an impressive number of gates, memory bits and hard-coded multiplier blocks. They are excellent for complex signal processing tasks.

Let's build one ourselves

This project started in 2013 when the author built a first prototype and started writing VHDL-code for DSP blocks. Compared to a DSP executing code from scratch to glory, it is in general easier to implement a complex signal processing chain in programmable logic since every step can be designed independently of the others. When done, all that remains to do is connect the signals between the different blocks. Explaining this development process is beyond the scope of this article, but all the code written and all other necessary information is freely available for anyone to study [1].

Though a useful and powerful combination, programmable logic and ADCs alone do not add up to a radio. The FPGA-DSP board needs to be complemented with a radio board to do the frequency conversion, as well as with analog filtering and amplification. Such a board will be presented in a future installment of this series. A control board for tuning the radio, adjusting the volume, etc. will be the subject of yet another article. With the three boards it is possible to build a complete radio with a front panel and in a suitable enclosure.

It should be noted that the system may also be controlled by a PC connected with a USB-to-serial adapter, or a Raspberry Pi communicating over its I^{2}C or serial port. A graphical user interface written in Python is available to demonstrate the functionality.

Circuit description

The FPGA-DSP board features a 24-bit ADC for sampling at an intermediate frequency in receive mode, a Cyclone IV FPGA for signal processing, a high-speed DAC for local oscillator and transmit signal generation, an audio interface for microphone and loudspeaker, a very stable high-precision oscillator and an I^{2}C or UART interface for a host controller (**Figure 1**).

The heart (or brain) of the FPGA-DSP board is the EP4CE10 Cyclone IV FPGA from Intel, formerly Altera (IC4). This chip can be configured for virtually any digital function. Governed by the firmware, at start-up its gates are configured by an external memory (IC5) known as the configuration memory.

For audio input and output there is a so-called CODEC (coder-decoder) — a TLV320AIC20K (IC3) — comprising two ADC/DAC channels with 16-bit reso-

PROJECT INFORMATION

FPGA DSP SDR
Radio Transmitter
Receiver Ham Radio

entry level
intermediate level
➡ expert level

8 hours approx.

SMD soldering tools, JTAG programmer (USB Blaster), Quartus Lite

£190 / €200 / $210 approx.

lution and a maximum sample rate of 25,000 samples per second (SPS). It has a built-in digital 8-kHz lowpass filter, besides a microphone amplifier and a speaker driver that can deliver up to 250 mW into 8 Ω. All audio inputs and outputs sport programmable gain/attenuation. The speaker gets connected to K2, the microphone to K3; all other channels available in IC3 are connected to K4, making them available for auxiliary equipment.
The main input of the board is through K1; it expects a differential signal, typically in the hundreds of kHz range. The signal passes a differential amplifier (IC1) and a discrete lowpass filter, to be digitized finally by 24-bit ADC IC2, a type AD7760. This chip has many power supply connections separated by supply voltage and/or passive filters. Its master clock is provided by the FPGA, level-shifted to 5 V by IC8.
There is also a fast two-channel digital-to-analog converter (DAC) on the board, a DAC5672 (IC6). Its two differential outputs A and B are converted with transformers to single-ended outputs; low-pass filtering is provided, allowing signals up to 50 MHz to be produced. The two DAC outputs are available at SMA-type coaxial connectors K8 and K9.
The board's main clock is generated by IC7, a 20-MHz Temperature-Compensated Crystal Oscillator (TCXO). The clock signal is fed to the FPGA which distributes it to the peripheral ICs. The frequency can be fine-tuned with potentiometer P1. Connector K5 provides an input for an external reference source — if one happens to be available. If so, the TCXO can be switched off by means of T1. The TCXO and the external reference signals are both guided to the FPGA through unbuffered inverters wired as analog amplifiers (IC9, IC10) and consequently may be of moderate voltage swing.
The clock is divided down in the FPGA to generate a low frequency signal for LED1 indicating that the clock is running and the FPGA is configured. The function of LED2 is reserved for future needs. The board's 5-volt supply voltage is connected to K10 from where it is distributed to four different Low Drop-Out (LDO) voltage regulators (IC11-14) to create the supply voltages — 1.2 V, 1.8 V, 2.5 V and 3.3 V — required for various components on the board.
The interface to the host controller consists of two pins intended for I²C or UART communication, selectable by solder jumper (JP1). They are available on K7, together with seven additional I/O lines intended typically for PTT (push-to-talk) and Morse key signals. These pins might also be used for I²S audio I/O.
Solder jumpers JP2 and JP3 do not have a function in the current FPGA firmware. K6 provides a JTAG interface for programming the FPGA and its configuration memory.

Features

- FPGA + DSP + Audio CODEC
- Narrowband software defined radio platform
- Can be used for radio operation at virtually any frequency between zero and many GHz
- Uses superheterodyne principle
- Low second IF higher than 0 Hz
- Receive (RX) and Transmit (TX) compatible
- Weaver SSB modulator/demodulator

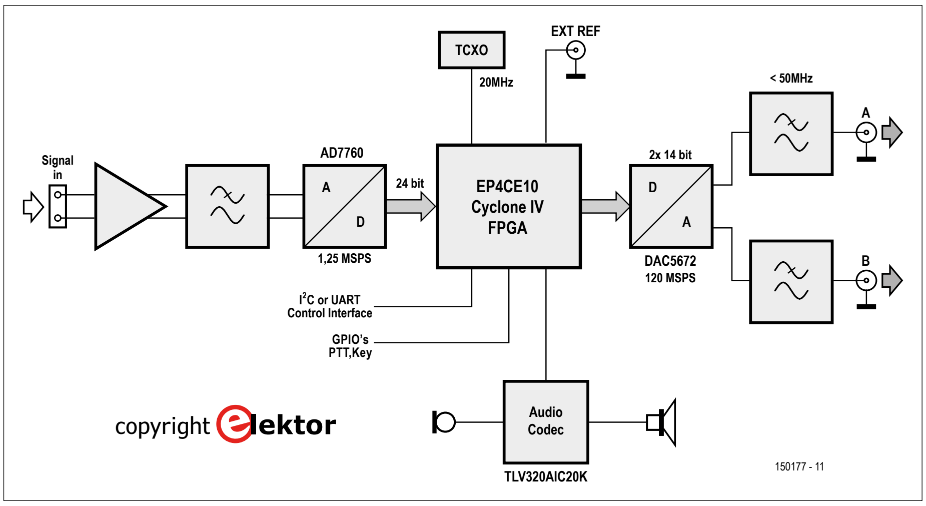


Figure 1. Block diagram of the FPGA-DSP board.

Radio topology

With a suitable radio board, the FPGA-DSP board can be used for radio operation at virtually any frequency between zero and many GHz. The typical applications, however, are for shortwave (under 30 MHz) and/or the 2-m amateur band (144-146/148 MHz).
For shortwave reception, a simple radio board could be constructed as shown in **Figure 3**. The antenna signal is lowpass filtered and amplified. By mixing it with a local oscillator (LO) signal from DAC A, the frequency of interest is converted to 45 MHz before it is passed through a crystal filter. This intermediate frequency (IF) signal is amplified before entering a second mixer where it is converted down to a second IF signal centered at

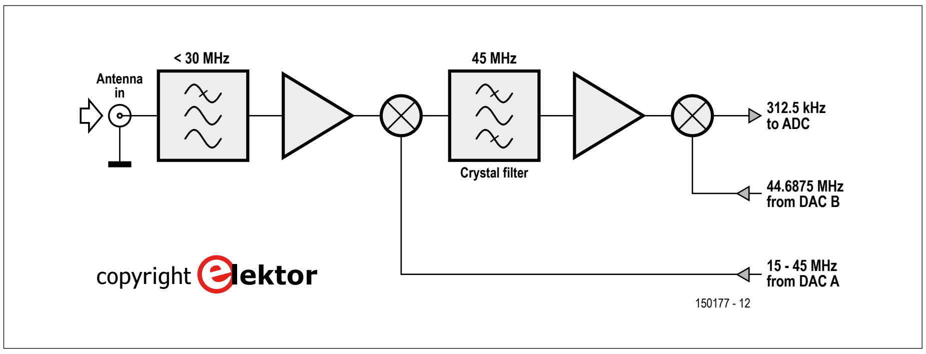


Figure 3. Block diagram of a radio add-on board.

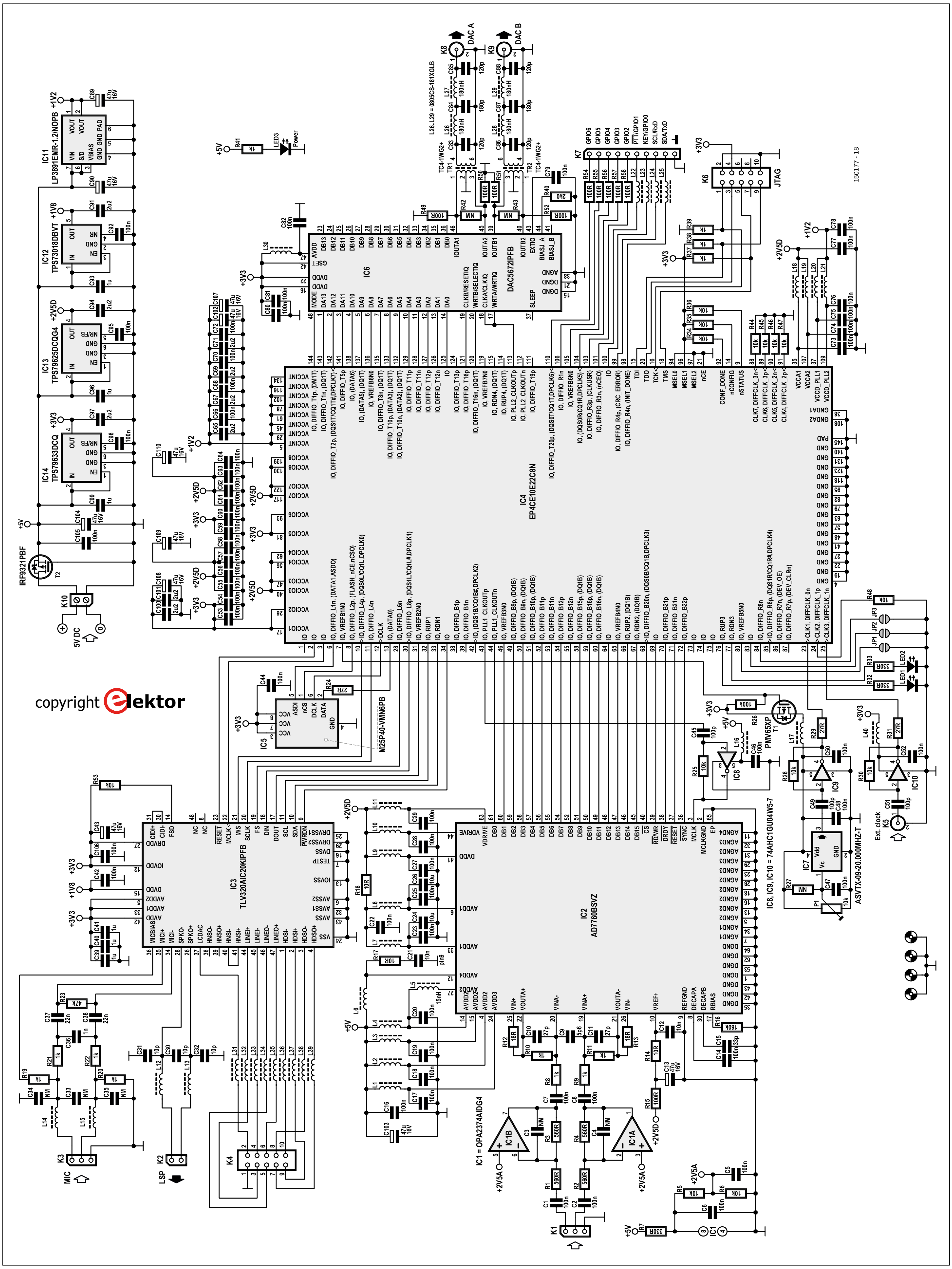


Figure 2. If you manage to look through the mist of passive filters, decoupling capacitors, data buses and supply lines, what remains is basically a four-chip schematic!

312.5 kHz (see below) with the help of a second LO signal provided by DAC B. The second IF signal is digitized by the ADC and fed into the FPGA where it is processed and ultimately demodulated to audio.
Compared to many other SDR-solutions this topology is advantageous for two reasons:

1. Double conversion superheterodyne principle.
The signal is mixed to a high-frequency first IF where it is filtered. Then it gets converted down to a low-frequency second IF and subsequently, sampled. This is advantageous since strong signals on frequencies other than the one of interest are filtered out by the narrow filter positioned early in the chain. This implies high selectivity, or high dynamic range, allowing the receiver to receive weak signals in the presence of strong ones.
The simpler SDR projects often use the 'zero-IF' principle instead, with only one mixer, no filter, and simple digitizing at audio frequencies around the carrier. The approach is limited in its ability to handle strong interferers, since mixing products of unwanted signals are likely to fall in the audio band where they will distort the signal you want to hear.
Fancier SDR solutions simply sample a large portion of the frequency spectrum ('direct sampling') and do the tuning in the digital domain. This method creates interesting opportunities for monitoring large chunks of the spectrum, but, when combined with a desire for good selectivity and/or dynamics, it places very strict requirements on the linearity of the input amplifier and resolution of the ADC in terms of effective bits. Fast sample rates and high resolutions usually imply high costs and power demands, often resulting in poor compromises.

2. The second IF is low but not zero.
This means that an interferer within the filter bandwidth (typically 15 kHz) cannot mix to produce a frequency interfering with the received signal. These kinds of mixing products will be sampled but digitally filtered out afterwards because they end up below the desired signal centered at 312.5 kHz. Since this frequency is not too high, it requires only a sample rate in the range of 1-2 MSPS for which one can find affordable 24-bit ADCs. These 24 bits result in a digital dynamic range of more than 100 dB and eliminate the need for gain control in the analog blocks, which, in turn, simplifies the receiver circuitry a lot.

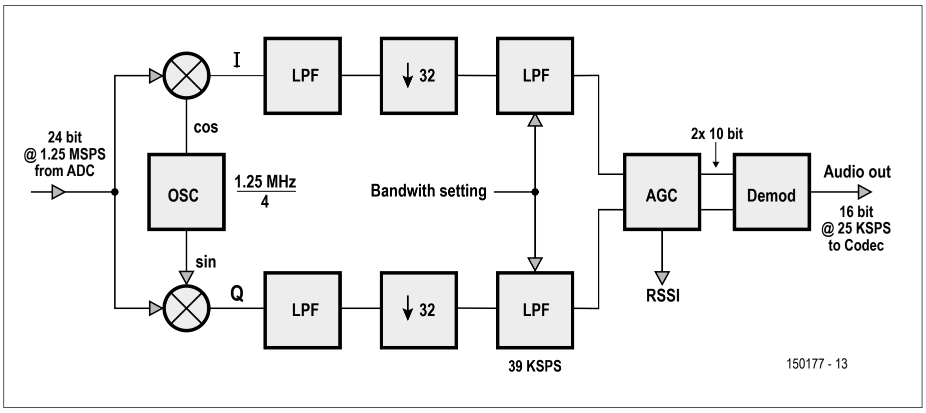


Figure 4. The signal processing chain implemented in the FPGA.

Digital signal processing — reception

It is the FPGA's job to process the sampled input signal. **Figure 4** shows a simplified diagram of the implemented blocks.
The ADC digitizes the input signal centered at 312.5 kHz at a rate of 1.25 million samples per second (MSPS). With 24 bits per sample at 1.25 MSPS there are 30 Mbits/s to process. This is a lot more than necessary and so the first digital processing steps are to down convert and down sample the input stream. By multiplying the signal with two sequences [0,1,0,−1,0,...], a "sine wave" at exactly a fourth of the sample rate, and [1,0,−1,0,1,...], a cosine (the second sequence is identical to the first, but shifted one position or 90°) the signal at 312.5 kHz is down-converted to a quadrature signal centered at 0 Hz. These signals are then fed to down-sampling low-pass filters that reduce the sample rates by 32 — simply by throwing away 31 out of 32 samples —, to provide outputs at sample rates of around 39 KSPS. These two down-sampled signals now each contain information from zero to about 10 kHz; combined, they represent the information that was within ±10 kHz of 312.5 kHz in the sampled input signal. In case you didn't guess it, the second IF of 312.5 kHz was chosen because it is a quarter of the 1.25-MHz sample rate. This allowed the implementation of the down converter in VHDL to be kept simple because multiplying by 1, 0, and −1 is easy (**Listing 1**). Other IF frequencies would have required a sine table and 24-bit multipliers.

Listing 1.
VHDL implementation of a down converter at ¼ of the sample rate.

```
if ns = 0 then
Ia(write_pointer) <= signed(Data_in);-- 1
Qa(write_pointer) <= to_signed(0,24);-- 0
ns := 1;
elsif ns = 1 then
Ia(write_pointer) <= to_signed(0,24);-- 0
Qa(write_pointer) <= signed(Data_in);-- 1
ns := 2;
elsif ns = 2 then
Ia(write_pointer) <= (not signed(Data_in)) + 1; -- -1
Qa(write_pointer) <= to_signed(0,24);-- 0
ns := 3;
elsif ns = 3 then
Ia(write_pointer) <= to_signed(0,24);-- 0
Qa(write_pointer) <= (not signed(Data_in)) + 1; -- -1
ns := 0;
end if;
```

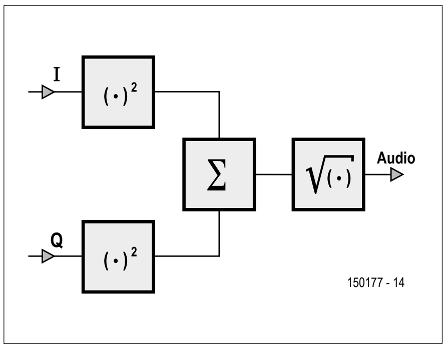


Figure 5. AM signal demodulation.

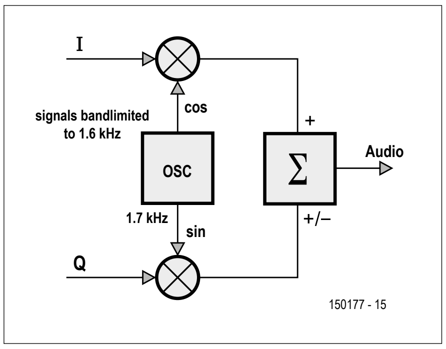


Figure 6. SSB and CW signal demodulation.

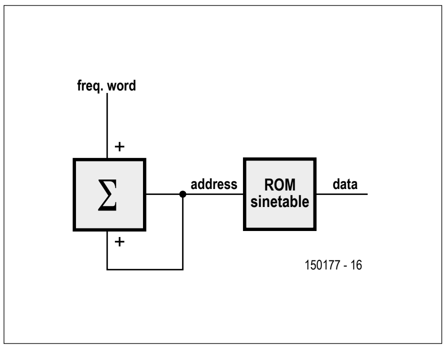
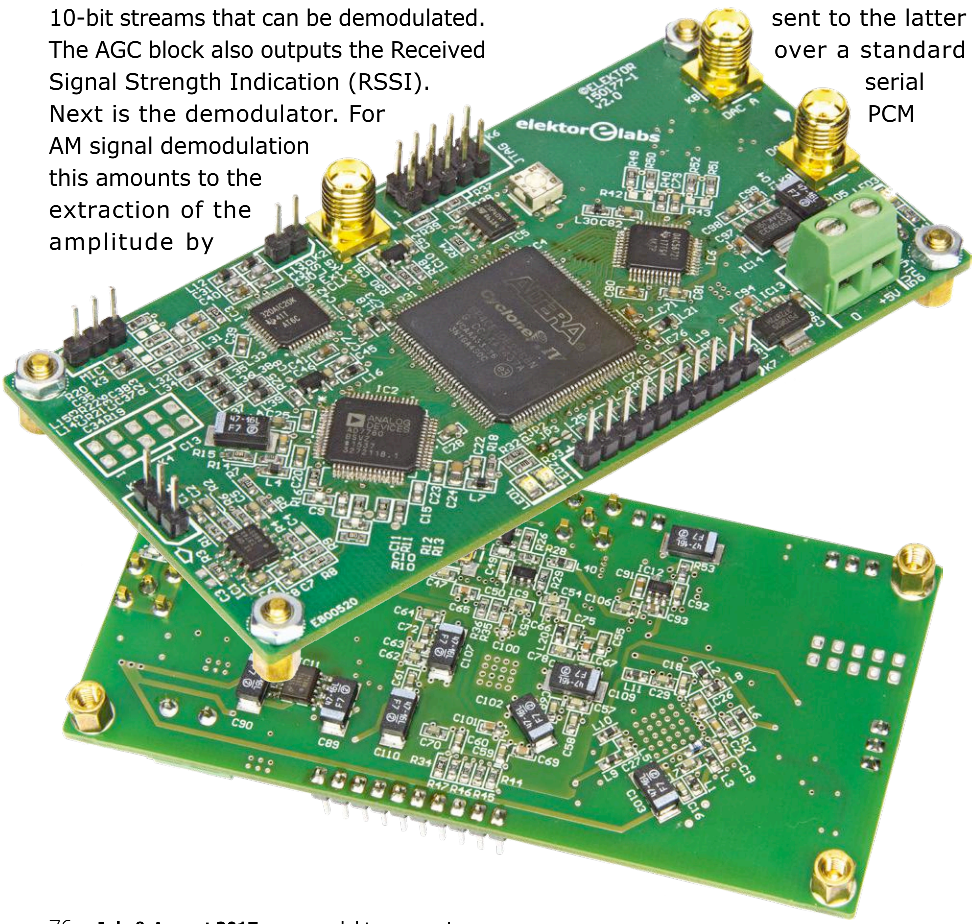


Figure 7. The basic operating principle of DDS. The block with the sigma symbol in it is the phase accumulator.

The next step is to lowpass-filter the two signals once more. This time the cutoff frequencies are adjustable in order to match the selected mode of operation. For AM, their cutoff frequencies are at 5 kHz for 10 kHz of information bandwidth. For SSB (Single-Side Band) telephony, they are set to around 1.6 kHz (3.2 kHz bandwidth), and for narrow band CW (Continuous Wave; Morse code) they filter at only a few hundred hertz.
Two times 24-bit resolution still yields a lot of data to process. To extract the information of interest, i.e. to demodulate, we do not need all these bits. The task of the next block, the Automatic Gain Control (AGC), therefore is to continuously monitor the signal strength and scale the signal so as to output only two 10-bit streams that can be demodulated. The AGC block also outputs the Received Signal Strength Indication (RSSI).
Next is the demodulator. For AM signal demodulation this amounts to the extraction of the amplitude by using the Pythagorean theorem for two orthogonal vectors: $\sqrt{(I^2+Q^2)}$, see **Figure 5**. For SSB (and CW) demodulation is unfortunately a bit more complicated. In this design Weaver's method was used [2][3]. The main idea (**Figure 6**) is to split the signal bandwidth in two and mix again with a locally generated low frequency quadrature signal in the middle of the band. Choosing between Upper Side Band (USB) or Lower Side Band (LSB) is then just a matter of sign manipulation in the final adder.
The demodulated audio finally passes a lowpass filter to remove any high-frequency components caused by distortion in the AGC or demodulator. It is then resampled to match the sampling rate of the audio CODEC and sent to the latter over a standard serial PCM interface. The audio CODEC provides programmable gain and can drive a speaker or headphone directly.
The local oscillator (LO) signals at the outputs of the two DACs are produced by two Direct Digital Synthesis (DDS) blocks inside the FPGA. Such a synthesizer consists of a so-called phase accumulator register whose value is increased by a fixed amount at every clock cycle of the 120-MHz DAC clock (which is derived from the internal PLL of the FPGA), thus the increase determines the frequency. The accumulated value is used as an address into a sine table; the value found at that address is sent to the DAC (**Figure 7**). The frequency word for DAC A, which forms the tunable LO, is 25-bits wide. This corresponds to a frequency step of $120 \times 10^6 / 2^{25} = 3.57$ Hz. In addition, there is a fine-tune mechanism, implemented with fractional values, giving a final resolution of 0.44 Hz.
For the second LO, which is a fixed frequency for conversion between the first IF and the second, the corresponding DDS is implemented with a lower resolution to save FPGA resources — it has a precision of about 38 Hz.



FPGA programming

The compiled firmware for this project is available as a 'JTAG indirect configuration' file (trx.jic) that first configures the FPGA as a bridge and then programs the configuration memory through the FPGA. This is done with the aid of a USB Blaster adapter that connects to a PC over USB to form a JTAG interface to the board. Such adapters can be found for little money on popular shopping websites. The software to use at the PC side is known as Quartus Lite and can be freely downloaded from the Intel/Altera website. If the complete development envi-

COMPONENT LIST

Resistors

Default: 1%, 100mW, 0603

R1,R2,R3,R4 = 560Ω
R5,R6,R25,R28,R30,R34,R35,R36,R44-R47,R48,R53 = 10kΩ
R7,R32,R33 = 330Ω
R8-R11,R19-R22,R37,R38,R39,R41 = 1kΩ
R12,R13 = 18Ω
R14,R17,R18 = 10Ω
R15,R49-R52,R54-R58 = 100Ω
R16 = 160kΩ
R23 = 47kΩ
R24,R29,R31 = 27Ω
R26 = 100kΩ
R27,R42,R43 = not fitted
R40 = 2kΩ
P1 = 10 kΩ trimpot, 4.5mm, SMD

Capacitors

Default: 5%, 0603

C1,C2,C5-C8,C14,C16-C19,C20, C22,C23,C25,C27,C28,C29, C42,C44,C46,C47,C48,C50,C52, C53-C64,C66,C68,C70, C72-C82,C92,C95,C98,C105,C106 = 100nF
C3,C4,C33,C34,C35 = not mounted
C9 = 5.6pF ±0.25pF
C10,C11 = 27pF, 1%
C12,C21 = 10nF
C13,C43,C89,C90,C102,C103,C104, C107-C110 = 47µF 16V, tantalum, 0.35Ω, SMD Case C
C15 = 33pF
C24,C26 = 10µF 6.3V, 0805
C30,C31,C32 = 10pF
C36 = 1nF
C37,C38 = 22nF
C39,C40,C41,C93,C96,C99 = 1µF, X7R
C45,C49,C51 = 100pF
C83,C85,C86,C88 = 120pF
C84,C87 = 180pF
C65,C67,C69,C71,C91,C94,C97,C100, C101 = 2.2µF, X7R

Inductors

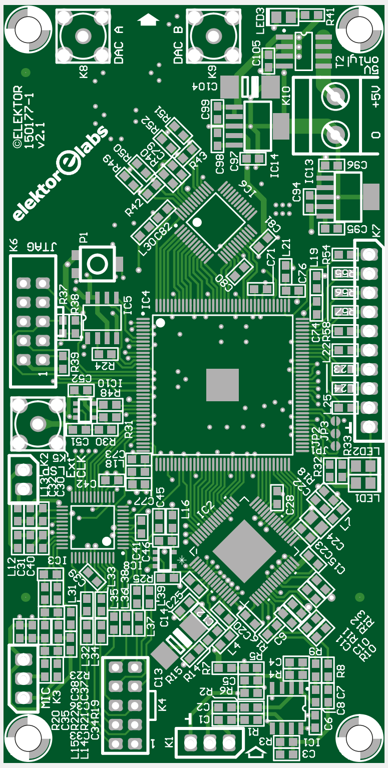
L1-L4,L6-L25,L30-L40 = 1kΩ @ 100MHz, 200mA, 0603
L5 = 15nH, 5%, 170mΩ, 700mA, f_{res} 4GHz
L26-L29 = 180nH, 2%, 640mΩ, 400mA, 0805
TR1,TR2 = TC4-1WG2+

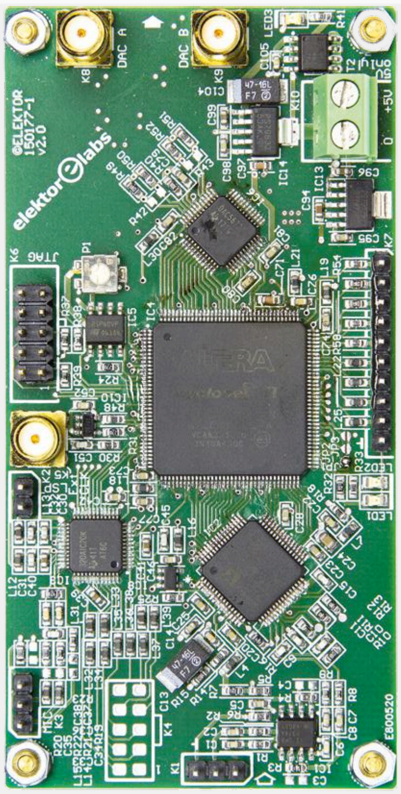
Semiconductors

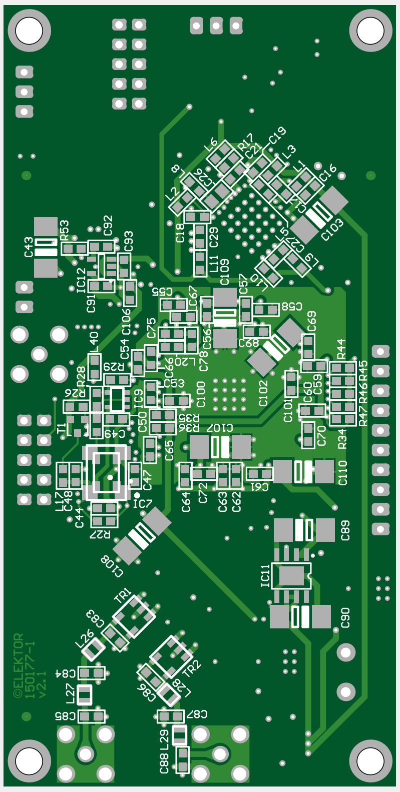
IC1 = OPA2374AIDG4
IC2 = AD7760BSVZ
IC3 = TLV320AIC20KIPFB
IC4 = EP4CE10E22C8N
IC5 = M25P40-VMN6PB
IC6 = DAC5672IPFB
IC7 = 20MHz crystal oscillator, adjustable, 5×3.2mm
IC8,IC9,IC10 = 74AHC1GU04W5-7
IC11 = LP3891EMR-1.2/NOPB
IC12 = TPS73018DBVT
IC13 = TPS79625DCQG4
IC14 = TPS79633DCQ
LED1,LED2,LED3 = green, 0805
T1 = PMV65XP
T2 = IRF9321PBF

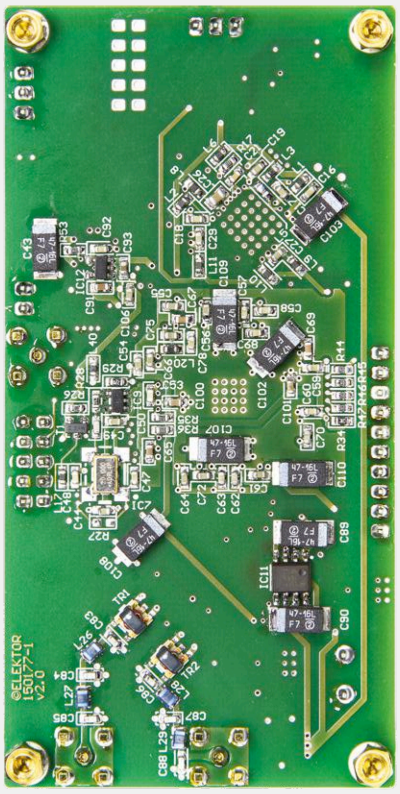
Miscellaneous

K1,K3 = 3-pin pinheader, 0.1" pitch
K2 = 2-pin pinheader, 0.1" pitch
K4 = 14-pin pinheader, 0.1" pitch
K5,K8,K9 = SMA straight jack, 50Ω
K6 = 10-pin (2x5) pinheader, 0.1" pitch
K7 = 10-pin pinheader, 0.1" pitch
K10 = 2-way PCB screw terminal block, 0.2" pitch
PCB # 150177-1









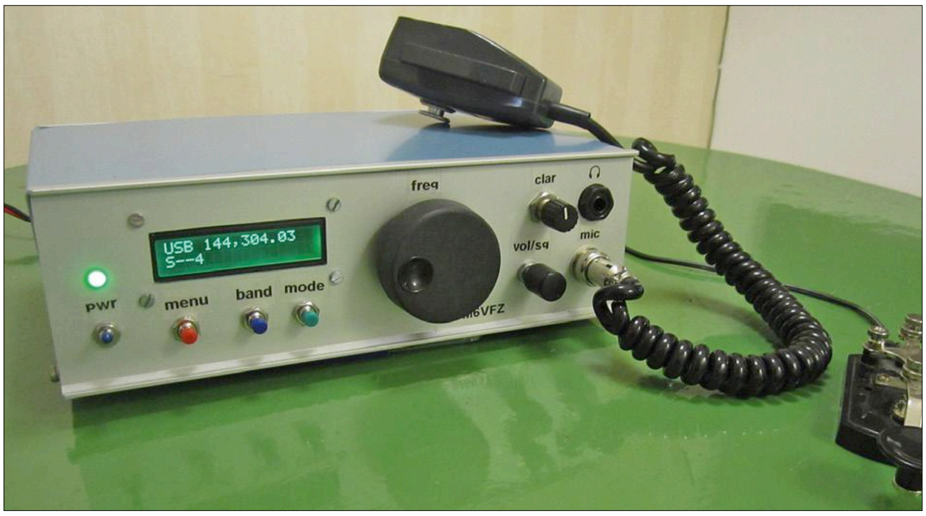


Figure 8. The prototype built by the author doing USB at 144 MHz.

Web Links

[1] www.elektormagazine.com/150177

[2] Donald K. Weaver Jr., "A Third Method of Generation and Detection of Single-Sideband Signals", www.h4.dion.ne.jp/~ja5fp/weaver.pdf

[3] Daniel Uppström, "Weavers metod för SSB", ESR Resonans 2/2014, http://resonans.esr.se/ESR_Resonans_2014_2.pdf (in Swedish)

[4] www.elektormagazine.com/labs/fpga-dsp-radio-for-narrow-band-communications-150177-i

[5] https://github.com/ast/dspsdr

ronment is not of interest there's always the option to download the applications needed for programming only. The Quartus suite is available for both Windows and Linux.

Control interface

At power on, the firmware gets loaded from the configuration memory, and programmed into the FPGA. Then the FPGA has to be initialized and controlled over I^{2}C or the serial port. The FPGA is much like a typical IC with a few registers that need to be set. Every data transmission involves five bytes being sent to the FPGA. The first two bits set the register address while the remaining 38 bits contain configuration data.

A few status signals can be read as well as the value of the received signal strength indication (RSSI). The controller typically polls this information a few times per second to update signal strength indication to the user.

A detailed description of the control interface, with full register map, is available together with the firmware and its source files in a GIT repository, accessible through the project web pages [1], [4], and [5].

In a standalone radio the controller is likely to be a small microcontroller board with a display and buttons (like the Elektor Platino). A suitable board will be described in a subsequent installment. For experiments, or when the need for a monitor and mouse/keyboard is not an obstacle, a Raspberry Pi should also make a flexible control interface. For the RPi (or any other Linux computer), an applet was written in Python using the GTK graphical framework, which makes the controls easily accessible. By default this applet connects to the FPGA board over the serial port. It also has an experimental socket mode to enable remote operation. The idea is to run a server on a Raspberry Pi at the location of the radio and connect to the Pi with the applet from a remote computer. With the large amount of man-made noise in urban areas, and with the limitations and regulations against installing large antennas in these locations, the option of remote operation becomes increasingly interesting for radio amateurs and shortwave listeners.

It should also be possible to route the audio from the FPGA directly to the PCM/I^{2}S ports of the Raspberry Pi in order to avoid the detour over analog audio when doing remote operation. This is yet to be tested.

How to download the applet, install necessary packages and connect the Raspberry Pi to the FPGA board is described in a document available with the source code.

Conclusion

The project presented in this article is a powerful building block for radio amateurs and experimenters who want to build their own radios without compromising on performance. At the same time it is hoped that this project will inspire the more digitally oriented readers to dive into the world of radio and signal processing.

The project has a lot of room for improvements and additions. A future installment (hopefully in the next edition) will present a radio board and discuss the transmission side of things; if enough resonance is observed, we may present more advanced radio boards.

The VHDL code for the FPGA is available as open-source on [5] and additions and improvements are welcome.

Finally, it should also be noted that the board is generic enough to be used as a development board for other FPGA-DSP applications.

Please visit the project web page for updates and additions. ⏮

(150177)