ARM Radio

a project for the ARM Microcontroller Design Contest

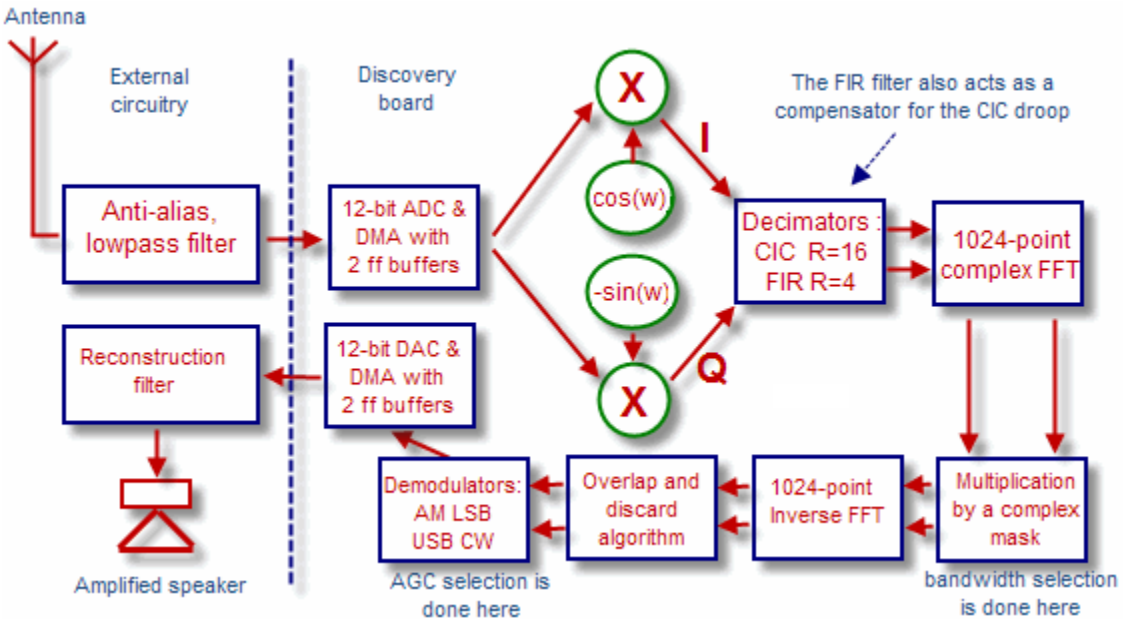
by Alberto di Bene, I2PHD

General Description

The goal of this project is to implement a SDR (Software Defined Radio) using the STM32F429 Discovery board, with a minimal amount of external hardware, using all the facilities offered both from the various devices of the chip itself and the middleware support (HAL, CMSIS, emWin, etc.) of the Keil compiler. This SDR is of the direct RF sampling variety, where the RF is immediately sampled by an ADC, then processed in numerical form with various DSP algorithms, all implemented taking advantage of the floating point core of the STM32F429 chip and the CMSIS library, and finally sent to the on-chip DAC to be transformed again in analog format, for reproduction by a speaker after being passed through a reconstruction filter.

Detailed Description

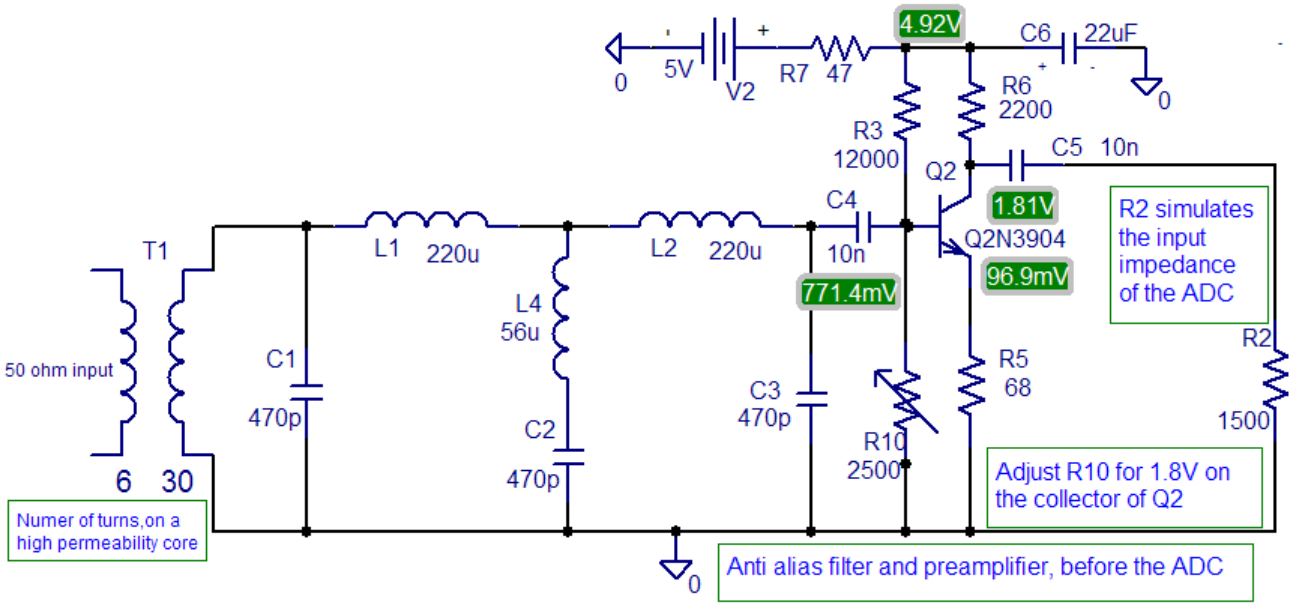
The block diagram of the flow of the signal inside the program is shown below. A detailed description of each block follows. In this block diagram the GUI (Graphical User Interface), i.e. the driving of the TFT display of the Discovery board is not shown, it will be done separately at the end of this document.



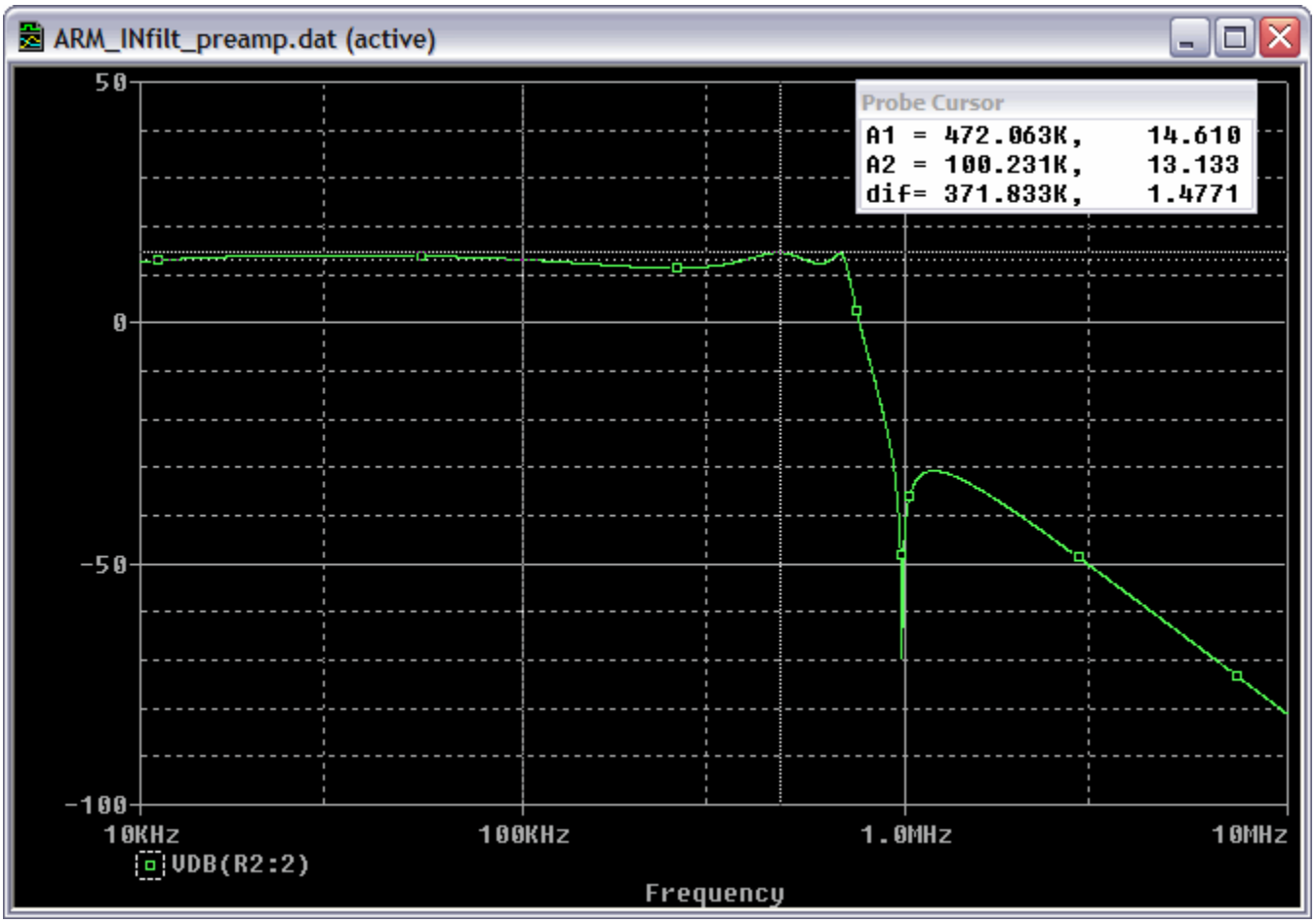
Anti-alias, lowpass filter

The need for an anti-alias filter in front of an ADC derives from the Shannon-Nyquist theorem, which, in its simplest form, states that no frequency components should be sent to an ADC with a value greater than half the sampling frequency. Otherwise aliasing will occur, i.e. the appearance of signals where they are not… in this project the sampling frequency is, nominally, 1.785714286 MHz (more on this later), so the low pass filter should attenuate substantially all the frequency components greater than 892.857143 kHz (again, nominal value). Given also that the ADC has an input impedance of a few kohms, compared with the usual 50 ohm of a properly dimensioned antenna, the filter does also an impedance transformation. Given also the not stellar performance of the ADCs of the ARM chip, whose ENOB is not particularly high, a small preamplifier of just over 10 dB of gain has been added.

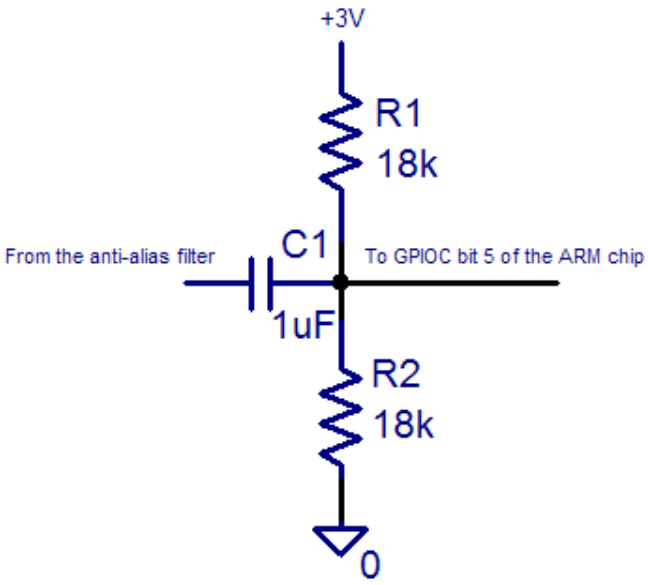
The following is the circuit adopted :



This is its Bode plot :



The incoming RF signal is centered on the zero value, and can have positive and negative values, while the ADC of the ARM chip has a digitizing range going from 0 to 3V. So it is necessary a simple level-shifting circuit, as follows :



Analog-to-Digital-Converter

The STM32F429ZIT6 ARM chip of the Discovery board sports three 12-bit ADCs, each capable of a sampling frequency of 2.4 MHz. They can be combined in interleaved mode, thus reaching 7.2 MHz. While this at first looked highly desirable, allowing to cover up to the 80-meter ham band, subsequent tests did show that the computing power of the ARM chip, given the complex algorithms for the digital processing used, did not allow to go so high in sampling frequency. The time spent in each task was measured with the technique of flipping a bit in one of the GPIO ports, and then using an oscilloscope to have a measure of how many microseconds the task duration was. This was done at various system clock speeds, trying to achieve the highest possible sampling rate.

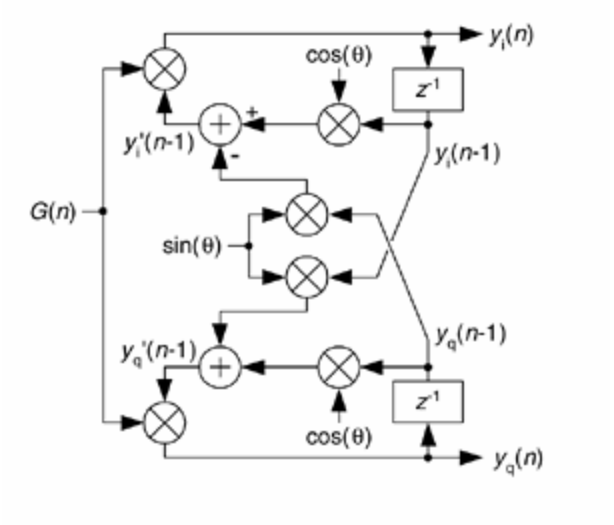
After some measurements, it was decided to use a clock frequency of 200 MHz for the ARM chip (it was tested also at 220 MHz for several days, with no glitches, but to stay on the safe side, 200 MHz was considered a good choice). Given the architecture of the ADC converters, to have the finest choice in the granularity of the sampling rate, the more logical choice seemed that of using two of the three ADCs in interleaved mode, with a delay of 14 cycles between the relative conversion starts. Shorter delays would have given a faster sampling rate, but the processor didn't have enough steam to process timely the data. That delay of 14 cycles gives a sampling frequency of :

Fs = (Fclock)/2/4/14 where Fclock = 200e6, giving Fs = 1.785714286 MHz (supposing the board xtal is spot on 8 MHz…)

With this sampling frequency, the coverage of the receiver goes from almost the DC (it is limited in the code at 8 kHz) up to the first part of the MW (Medium Waves), covering all the LF and the NDB band. The ADCs rely on the DMA controller to store the sampled data in two flip-flop buffers, each of 512 entries. While one buffer is filled by the DMA, the other buffer is processed, and so on. The two ADCs sample with an amplitude resolution of 12 bits, in a short word format. This format is converted to floating point, normalized to 1, for the subsequent processing.

NCO (Numerically Controlled Oscillator)

The NCO is used to bring at zero IF the reception frequency desired, using two complex mixers, generating the I/Q components of the analytic signal at zero IF. A first implementation of the NCO was attempted with a 16384-entry sine table, scanned at various speeds, more or less like a DDS works. That worked, but had the inconvenient that at some frequencies the granularity was rather coarse, not permitting a fine tuning. This could have been alleviated by using a larger sine table, but at the expenses of an excessive use of memory. At this point the NCO was implemented using a quadrature coupled complex oscillator, practically a double IIR filter brought on the verge of oscillation, complete with level stabilization code.



This worked quite well, albeit requiring some more CPU cycles than the sine table approach… but applying some optimization to the code, that was acceptable. The code for the NCO is in the routine *SDR_ComputeLO* in the module SDR_math.c

The algorithm for the quadrature coupled complex oscillator was lifted from the Richard Lyons' book mentioned in the Decimators paragraph, just below.

Complex mixers

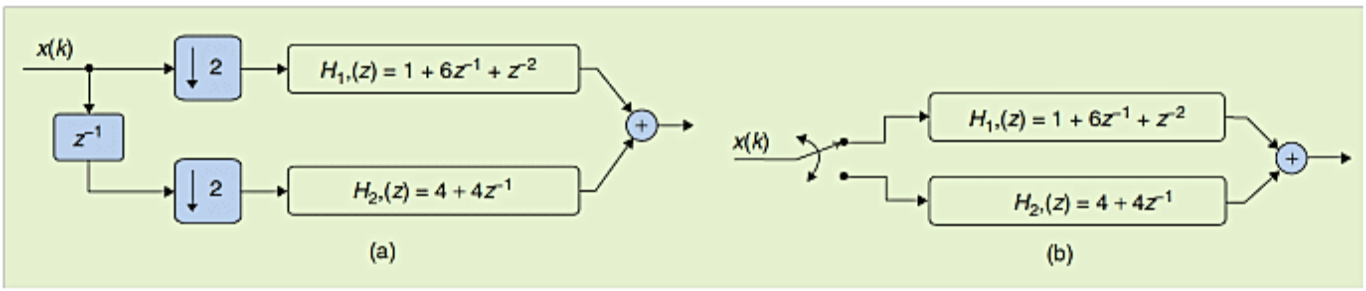
In the digital world it is possible to implement that ideal mixer that in the analog world is only a chimera… an ideal mixer is… a multiplication, nothing more, nothing less. The signal coming from the ADC is real, while the one provided by the NCO is complex, so the output of the two multipliers is a complex signal, conventionally designated as I and Q components. This output signal is centered at zero IF, but still has the original sampling frequency of the ADCs. So it must be downsampled to bring it to a more manageable value for the baseband processing.

Decimators

It was decided to first downsample by a factor 16 using a 4th order CIC, then by another factor 4, using a 64-tap FIR filter, computed so to also compensate for the typical droop of a CIC filter, which, in this case, was really very small, but it was nevertheless compensated, given that it did not require any additional CPU cycles.

The CIC filter was not designed according to the original 1981 Hogenauer paper, but, following what Richard Lyons writes in his excellent book "Understanding digital signal processing", a polyphase decomposition approach was used, that has the advantage of eliminating the integrating part of the CIC, thus allowing the use of floating point, which is rather fast on the Cortex M4F chip, given also the excellent optimization of the C code performed by the Keil compiler.

This is an example of how the implementation of one section of a CIC implemented with the polyphase decomposition works :

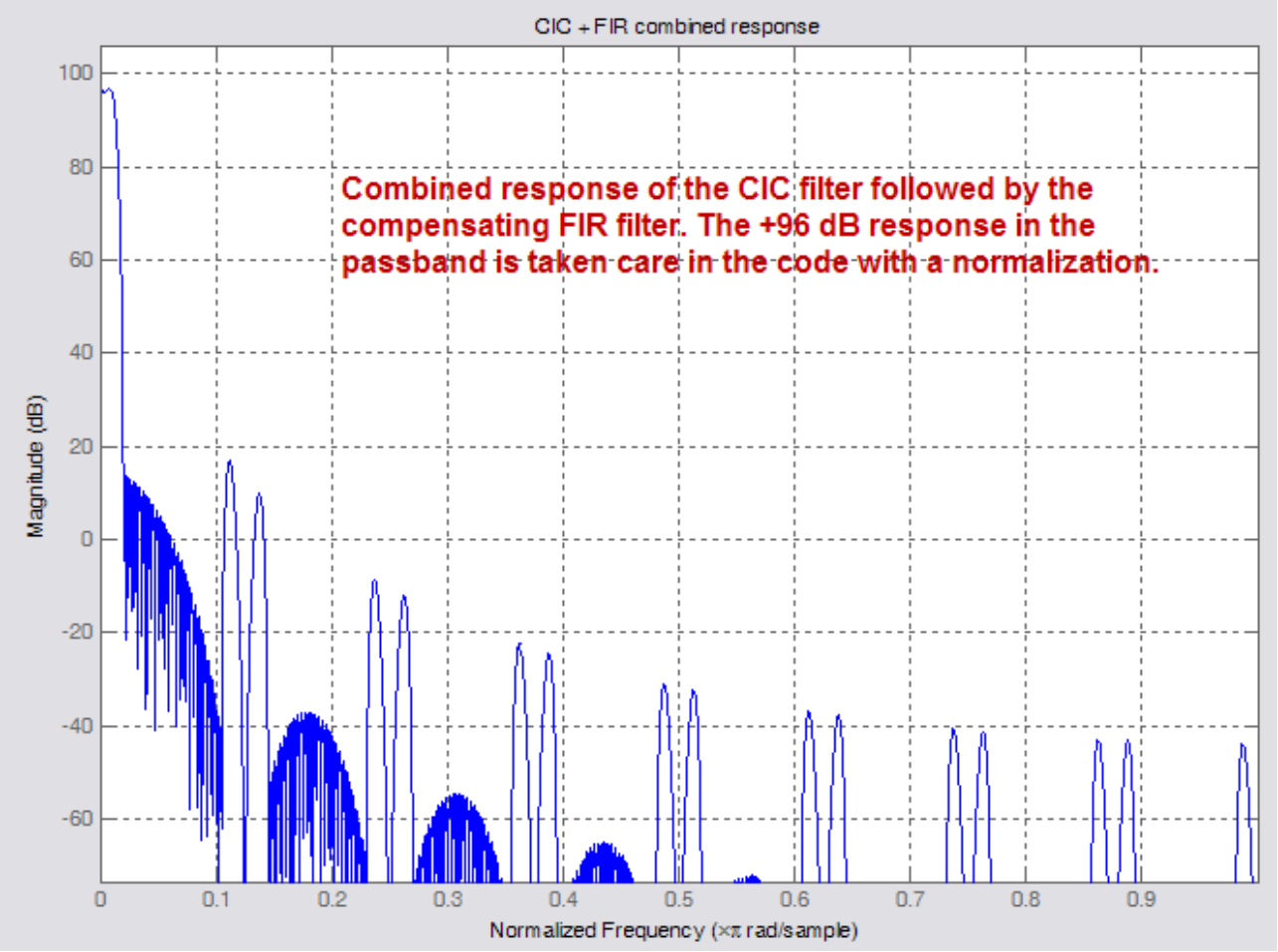


The CIC is followed by a 64-tap FIR filter, computed with the Remez exchange, equiripple algorithm. The combined response of the CIC and the FIR is reported in the following figure. The decimation by 64 gives a processing gain that can be computed with this formula :

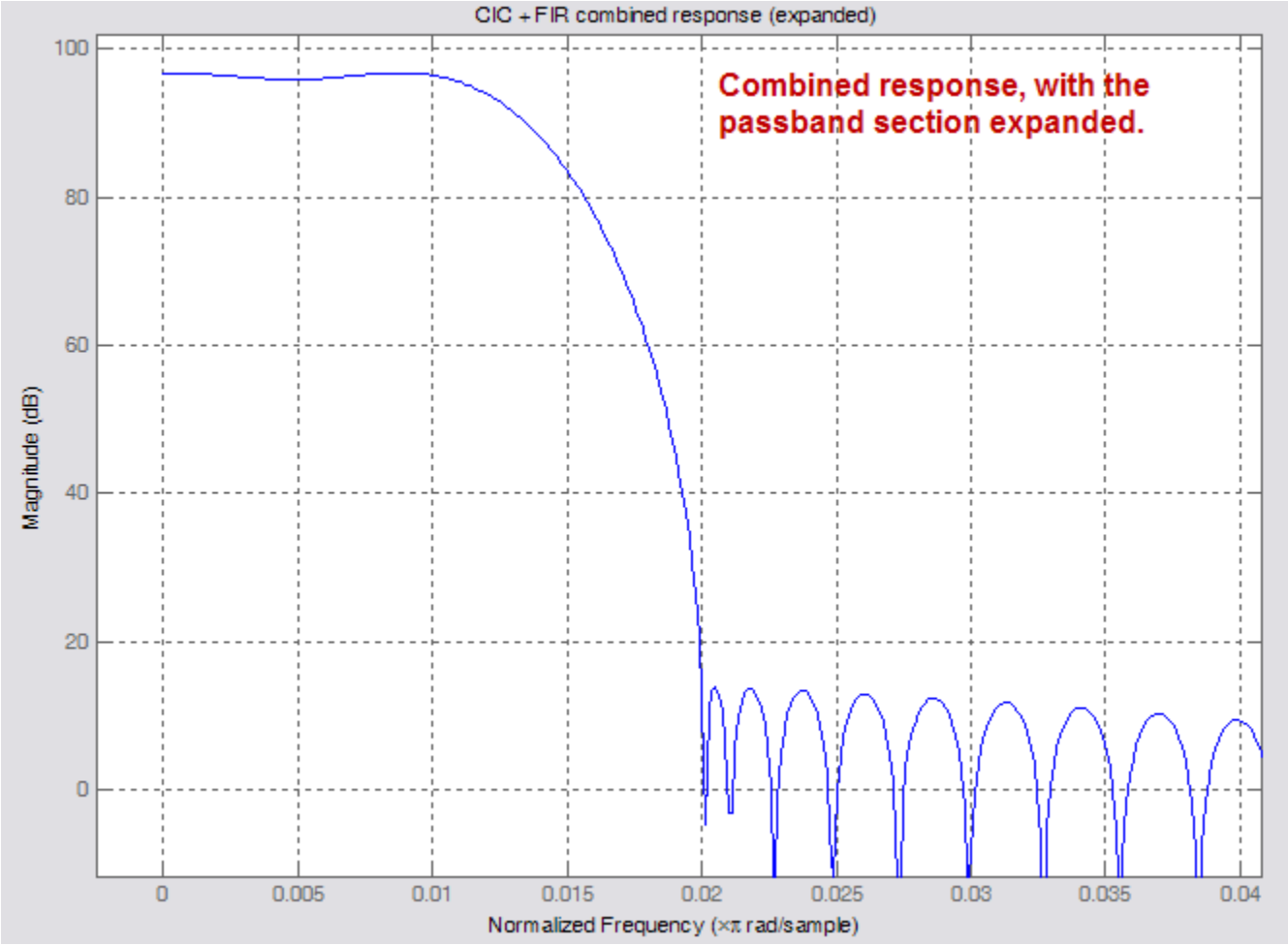
$$SNR = (6.02N + 1.76) + 10*\log10(Fs/Fd)\ dB$$

Where N is the number of bits of the ADC, Fs/Fd the decimation ratio, and SNR the Signal-to-Noise ratio that can be expected. Putting numbers in the formula, we have a theoretical SNR of about 92 dB. This of course if the ADC had an ENOB equal to the number of bits used for the digitization, which unfortunately is not always the case…

Combined response of the CIC filter + the FIR filter



This is an expanded view of the passband :



The FIR was implemented using the optimized CMSIS routine *arm_fir_decimate_f32* which proved quite effective as far as the optimization level was concerned.

The CIC code can be found in the routine *DMA2_Stream0_IRQHandler* in the SDR_func.c module. That routine runs at the highest priority (NVIC level 0), being the routine that serves the DMA2, Stream 0, interrupts, those generated when one of the two flip-flop buffers is filled.

The FIR code can be found in the routine *EXTI1_IRQHandler* in the SDR_func.c module. That routine serves the software interrupt generated by the *DMA2_Stream0_IRQHandler* routine, every time a CIC-downsampled buffer is ready. Its priority is just below that of the previous routine.

Bandpass Filtering

Bandpass filtering is performed with the fast convolution algorithm, which consists in computing the FFT of the input signal, multiplying it with a selectable mask, and then computing back an inverse FFT. This is made possible by the fact that a convolution in the time domain is equivalent to a multiplication in the frequency domain. So, if a suitable kernel for a FIR filter is computed, its Fourier transform can be used as a mask to be multiplied by the Fourier transform of the input signal. Of course, as the elementary DSP theory preaches, simply doing that would result in a circular convolution, not in the desired

linear one. This latter can be had by using one of two methods, namely the overlap-and-add, or the overlap-and-save, which I prefer to call overlap-and-discard, given that only a part of the computed inverse FFT is retained.

Describing in details the two methods is inappropriate for a document like this, a very good description can be found in the aforementioned Richard Lyons' book.

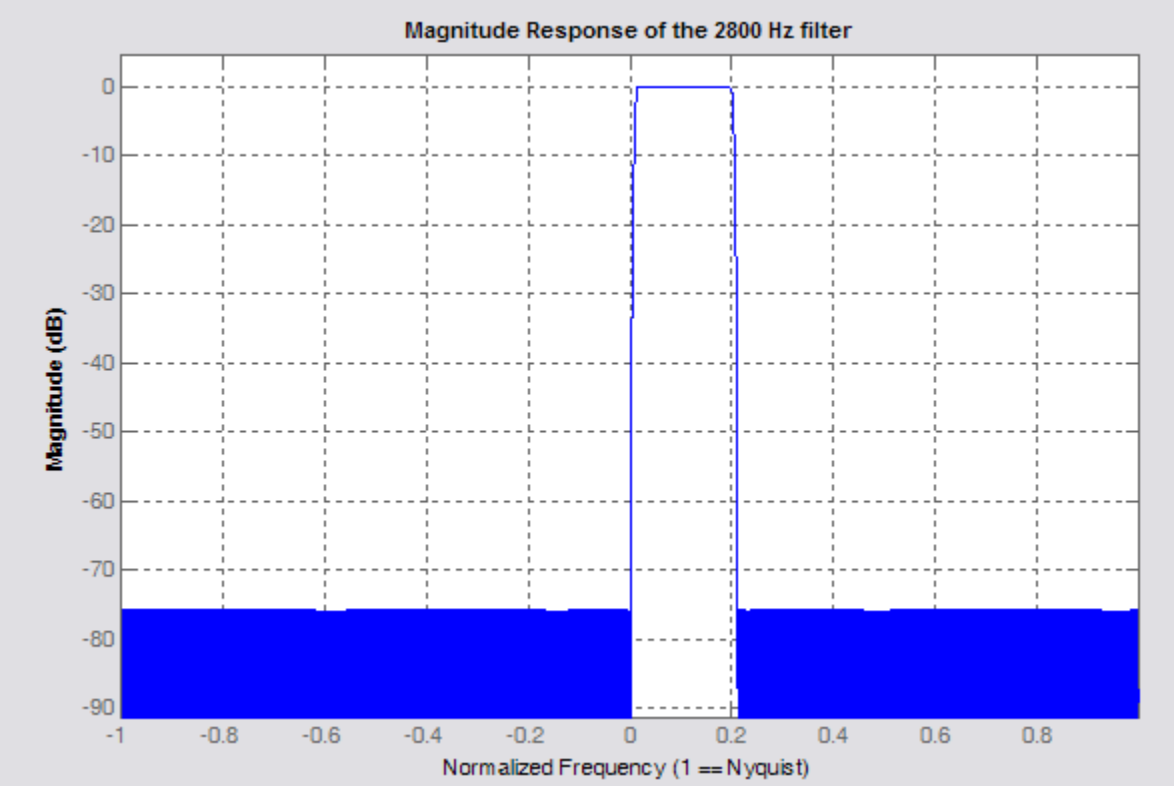
The latest release of the CMSIS library has a very good and fast implementation of the floating point complex FFT, using a mixed radix approach, so it was used both for the direct and inverse FFT, with a size of 1024 points. The masks were computed using Matlab m-functions, written ad-hoc. First a 513-tap kernel is computed, with the passband as in the following table, using the firpm primitive of Matlab, and then it is expanded up to a length of 1024, adding zeros. At this point a 1024-point FFT is computed, with its real and imaginary parts written to an external file, to be then imported as an .h file in the source of the application.

Bandwidth table

	Narrow	Wide
AM	3500 Hz	5500 Hz
SSB	2200 Hz	2800 Hz
CW	300 Hz	600 Hz

Additionally, in CW mode and with the Narrow bandwidth selected, the signal passes through a double IIR filter, implemented with a complex resonator with two conjugates poles placed inside the unitary circle at a radius of 0.987 from the origin, and with an anomaly corresponding to the CW pitch, namely 650 Hz. This further cleans the CW signal.

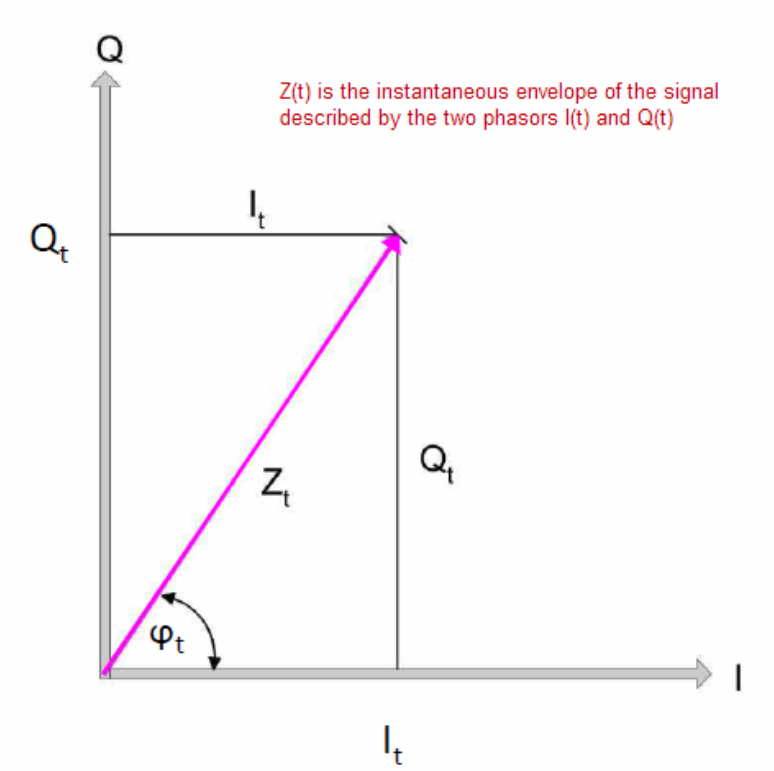
As just an example of the selectivity achieved with the fast convolution method, the following is a Matlab plot of the 2800 Hz SSB filter :



Demodulators

After the filtering, the signal is sent to the demodulator specific for the chosen mode. The CW demodulator is the same as the SSB demodulator, the only difference being that the frequency of the NCO is offset by the amount of the CW pitch (650 Hz) with respect to the tuned frequency.

Having the signal in analytic form (I and Q) makes the demodulation more easy. For AM demodulation what is needed is the envelope of the signal. This can be easily computed using the Pythagorean theorem :



Of course, there is a residual DC component which represents the magnitude of the AM carrier, and which must be eliminated. This can be done with this simple IIR filter :

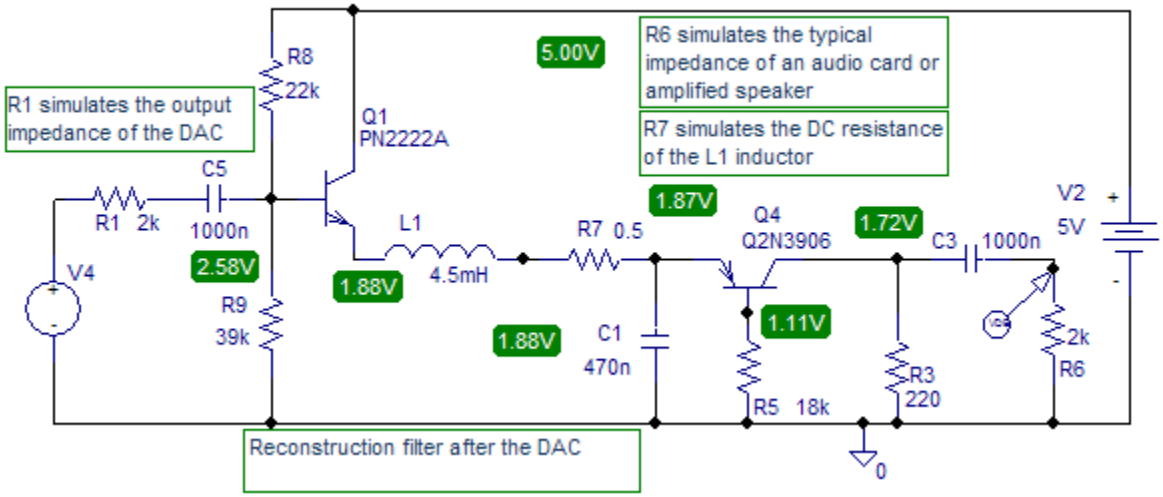
```
// DC removal filter -----------------------
  w = audiotmp + wold * 0.96f;
  fAudio[j++] = w - wold;
  wold = w;
// ----------------------------------------
```

The SSB demodulation is a misnomer, as nothing must be demodulated… the audio is just the real part of the complex signal resulting from the overlap-and-discard algorithm. For the LSB mode, a spectral inversion is done after the direct FFT, before the multiplication by the convolution mask.

The resulting audio signal is then subject to AGC, with a hang time and a decay time depending on the user choice (Fast or Slow). At this point the processing is almost done, the signal is converted back from normalized floating point format to 16-bit short integer words, ready to be sent to the DAC. The DAC is clocked by the Timer 6 of the ARM chip, with a frequency derived from the system clock, thus synchronous with the acquisition clock. Also for the DAC the services of the DMA controller are used, namely the Stream 6 of DMA 1, again with two flip-flop buffers. One is filled by the code while the other is emptied by the DMA controller, and so on.

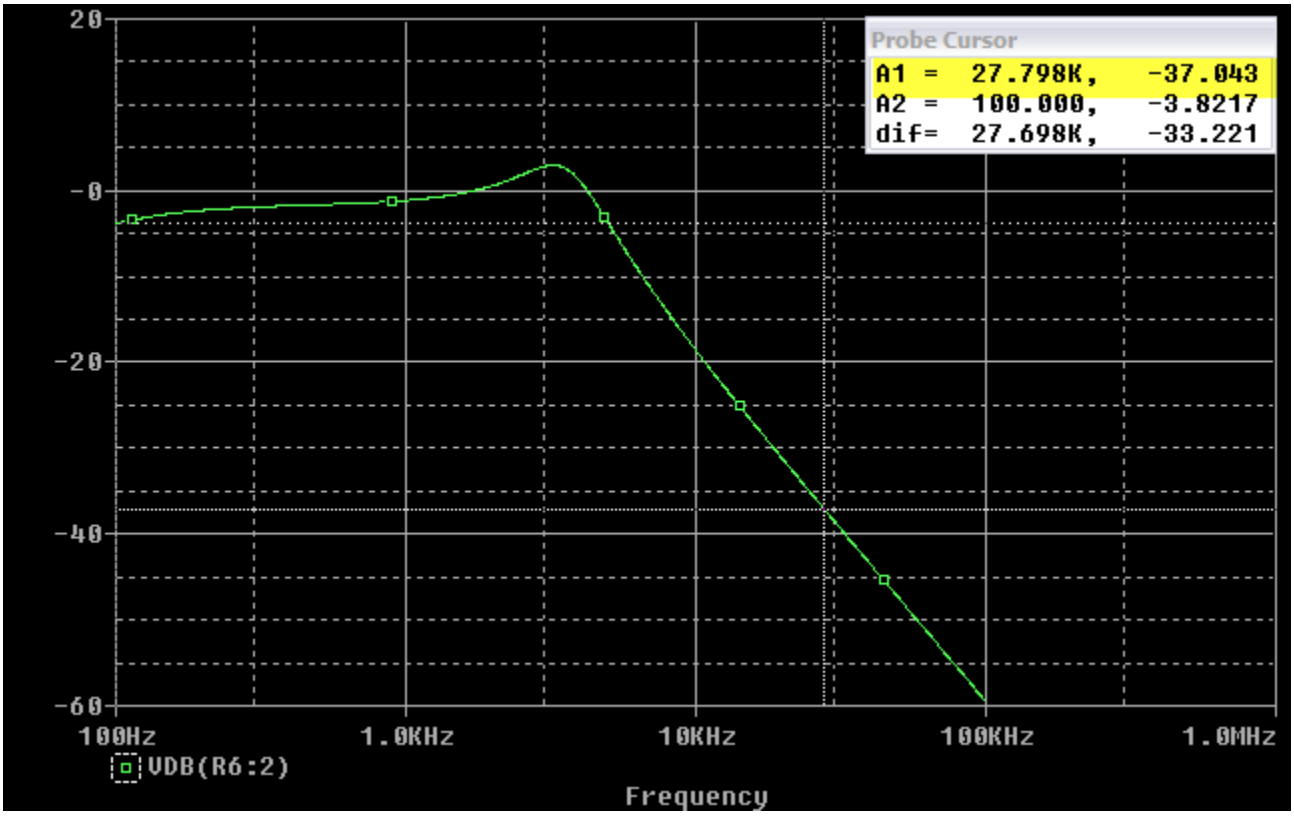
Reconstruction filter

The output of the DAC must be filtered, again according to the Shannon-Nyquist theorem, with what is commonly known as a reconstruction filter, whose task is essentially that of filtering out the DAC sampling frequency. A first attempt using a passive implementation showed that the value of the inductors needed, considered the output impedance level of the DAC, was so high to make them of a not easy availability, making difficult to replicate the circuit. At this point an active approach was chosen, with a couple of transistors that allowed working at much lower system impedance. This is the schematic :



In the actual circuit C5 is connected to bit 5 of GPIOA, which is the output of channel 2 of the DAC.

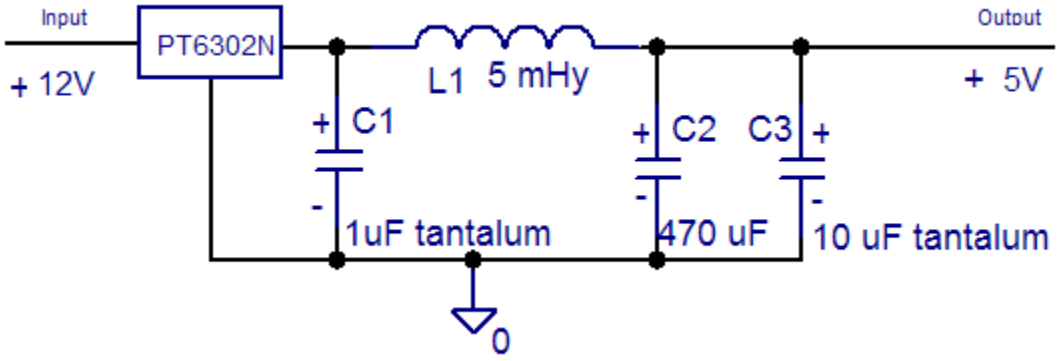
This is the bode plot of that circuit. The sampling frequency is attenuated more than 37 dB, a value deemed adequate.



The audio signal is now ready, and it is present on a jack for connection to an amplified speaker.

Power supply

The Discovery board accepts both 5V or 3.3V as supply voltage. In this realization, a small PT6302N module was used, a positive step-down switching regulator. It accepts input voltages from 9 to 30V, with a 5V output, with 3A max current. Being it a switching regulator, its output was examined with an oscilloscope and found to be not completely clean from the switching frequency. It was then decided to filter the output with this simple network :

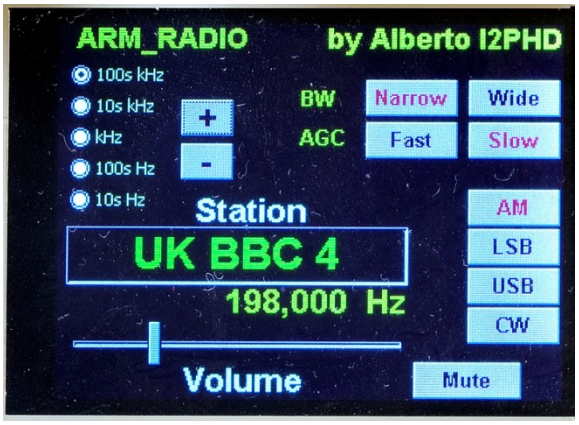


The 5V output, observed with an oscilloscope, resulted very clean.

GUI (Graphical User Interface)

The user interacts with the program using the touch screen, and the on-board User button. The panel was designed using the middleware component emWin of the Keil uVision environment (the GUIBuilder utility), and managed with the emWin primitives (GUI_Exec, GUI_Touch_Exec, etc. etc.). The emWin Exec functions, those who refresh the screen and take care of the touch gesture of the user, are executed at the lowest level of priority, the idle priority, when both the input acquisition task and the signal processing task had completed their chunks of work, waiting for the next buffer to arrive. That notwithstanding, the responsiveness is excellent, no slugginess is perceived.

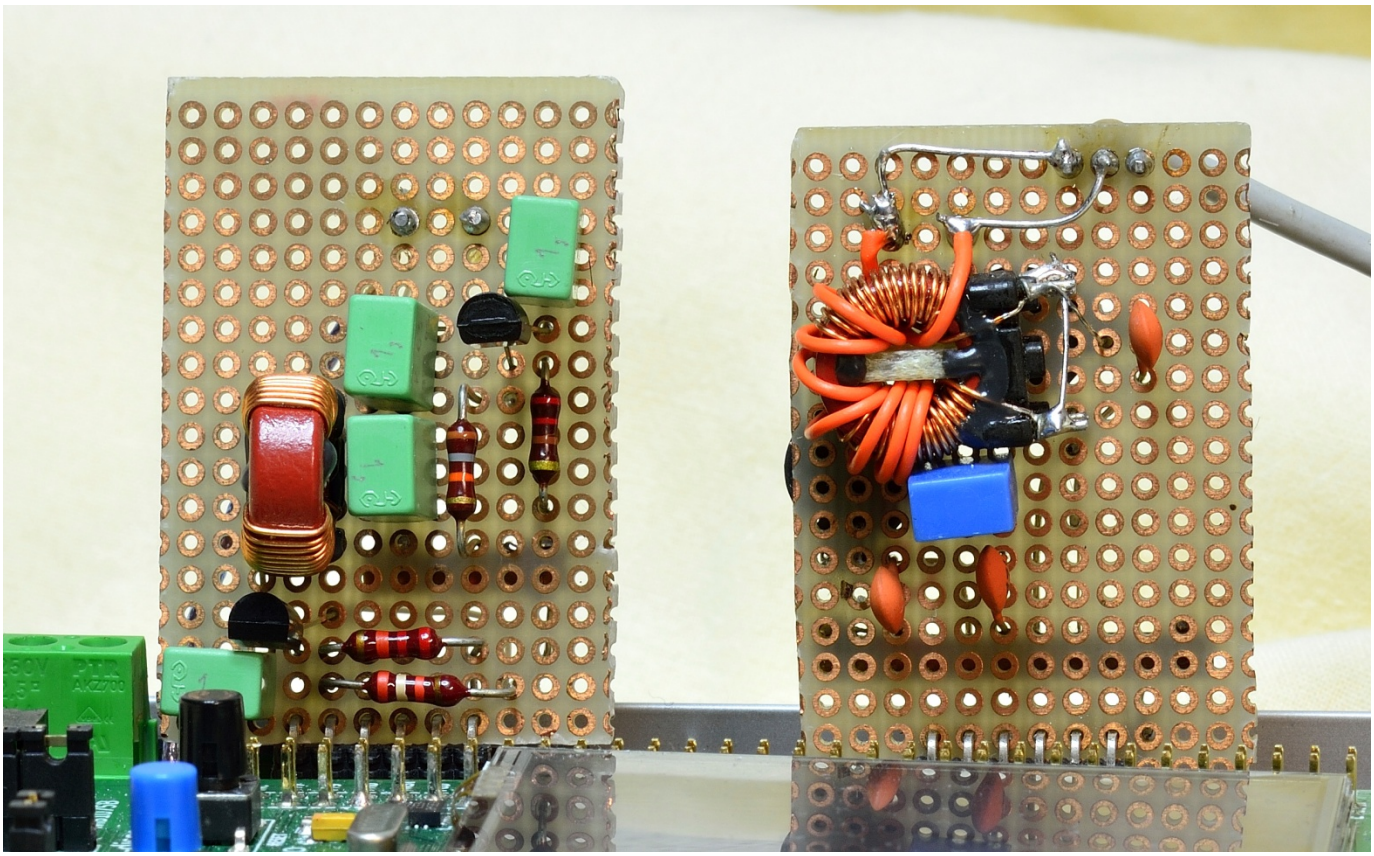
The User button, present on the Discovery board, is used to cycle among a few station presets, whose name, mode, bandwidth and AGC settings are in a table contained in a .h module, easily modifiable. This is a sample of the appearance of the screen, tuned to BBC 4 at 198 kHz :



By touching the on-screen buttons, it is possible to change the tuned frequency (+ or -) by the amount selected with the radio buttons. Also the bandwidth, the AGC speed, the demodulation mode and the volume are all settable with the touch-enabled buttons.

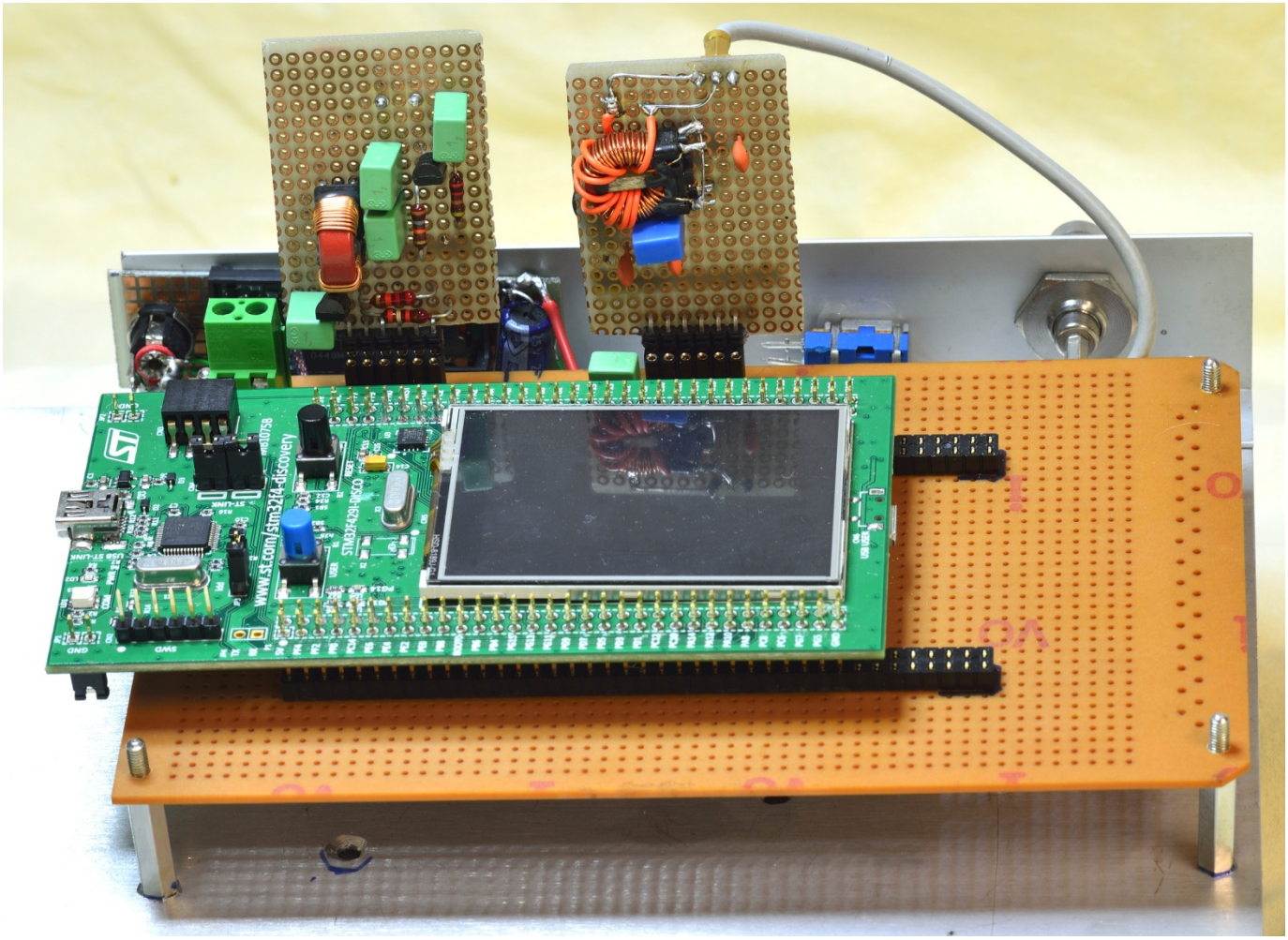
A few photos of the realization

The realization is done in a very amateurish fashion, the filter are hand-soldered on a couple of breadboard circuits, this is a photo of them :

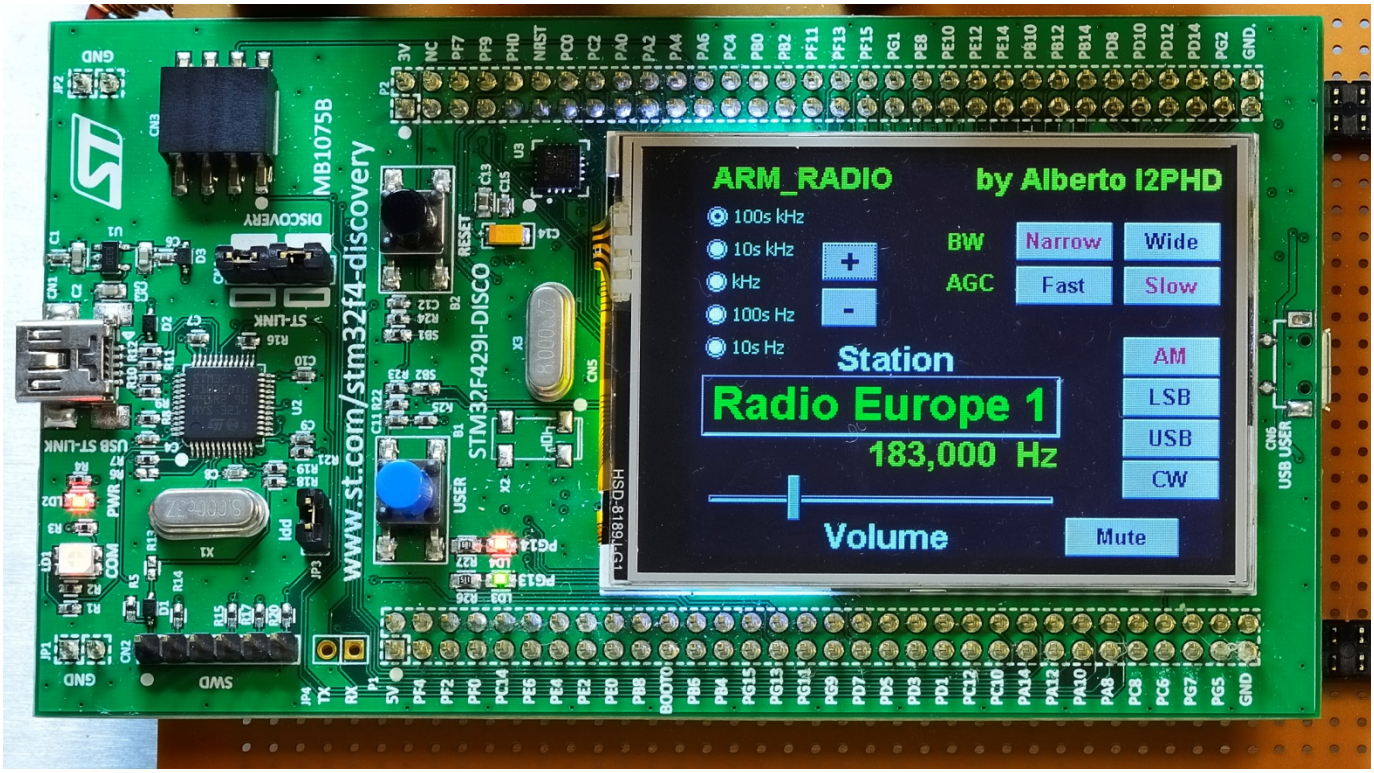


Note : the photo of the right board, that with the input anti alias filter, is of the initial version, that did not have the preamplifier, which was added at a later time

Here it is possible to see the entire project, the Discovery board with the two breadboard circuits, the input and output filters:



And finally this is a photo taken when the program was running, tuned to the Radio Europe 1 station:



There is a Web page devoted to this project, at this URL :

http://armradio.weaksignals.com

Going there, it is possible to download the C source code of the program. Make sure to read the ReadMeFirst.txt file contained in the ZIP archive. Also a couple of MP3 audio files representing the reception possible with this project can be downloaded from that page.

Thanks for reading this.

Alberto di Bene I2PHD, i2phd (at) weaksignals.com