# Standalone Teensy SDR Receiver

Frank DD4WH.swl [ät] gmail.com 18th Dec 2015

(basic design by Rich HeslipVE3MKC with software improvements by Loftur VE2LJX)

- Reception from 16kHz to 36MHz
- LSB, USB, DSB, AM & pseudo StereoAM demodulation of signals from -120dBm (S0) to -13dBm (S9+60)
- Passband tuning and automatic snap tune (quasi-SAM)
- Software adjustable gain and phase for I and Q channels→ mirror rejection more than 45dB (sophisticated guess ;-))
- Frequency spectrum display with 44.1kHz bandwidth (by using complex FFT)
- Quadrature NCOs and complex multiplication used for translating from IF to baseband → all filters work on the baseband
- Graphical S-meter calibrated and working accurately independently of hardware and software gain settings
- Software filter chain: FIR (66 taps)/HILBERT(100 taps) plus IIR with fixed coefficients (8th order = 4-stage biquad with inverse Chebychev response = 60dB at 1.4fc !) plus IIR (8th order biquad = 4-stage biquad with Linkwitz-Riley response = 48dB / octave)
- Smoothly adjustable filter bandwidth (400 Hz – 8kHz) of IIR filter, FIR filters and fixed coefficient IIR filters are switched in automatically with corresponding bandwidths and predefined coefficients (Hilbert +45/-45 degrees type for LSB/USB, FIR without phase shift for AM)
- FIR and IIR filters with the following precalculated coefficients: 1.8kHz, 2.6kHz, 3.6kHz, 4.4kHz, 6.5kHz & 8kHz bandwidth
- Postprocessor tone control for bass and treble
- Audio recorder for SDR radio audio (up to 255 files) with playback option
- Built-in Real Time Clock backed up with battery
- Manually controlled high performance preselector (similar to BCC Preselektor)
- Single balanced quadrature sampling detector (QSD) (design and PCB by Joris KTH rf design)
- Local Oscillator Si5351 with Johnson Counter (74AC74) to divide f by 4 for QSD and simultaneously improve phasenoise of LO
- Sample and hold FST3253 with 33nF sampling capacitors
- Amplification with ultra low noise op amp LT6131
- Full user control over gain: Volume pot controls analog gain of headphone amp (in the codec SGTL5000) and analog gain of amp on front of ADC controllable by encoder (menu RFGAIN). Digital gain will be used for AGC, not yet implemented
- Fixed point DSP with Teensy 3.1 Microcontroller
- 12V operating voltage, ca. 120 - 150mA with low noise switching regulator
- Cost ca. 100 € (130€ with optical encoder)

# Teensy SDR Rx DD4WH software audio path – 25.11.2015

**Inspired by Rich VE3MKC, Loftur VE2LJX, Clint KA7OEI and kpc – Thanks!**

IQ Amplitude and phase calibration in audio_in_I & audio_in_Q

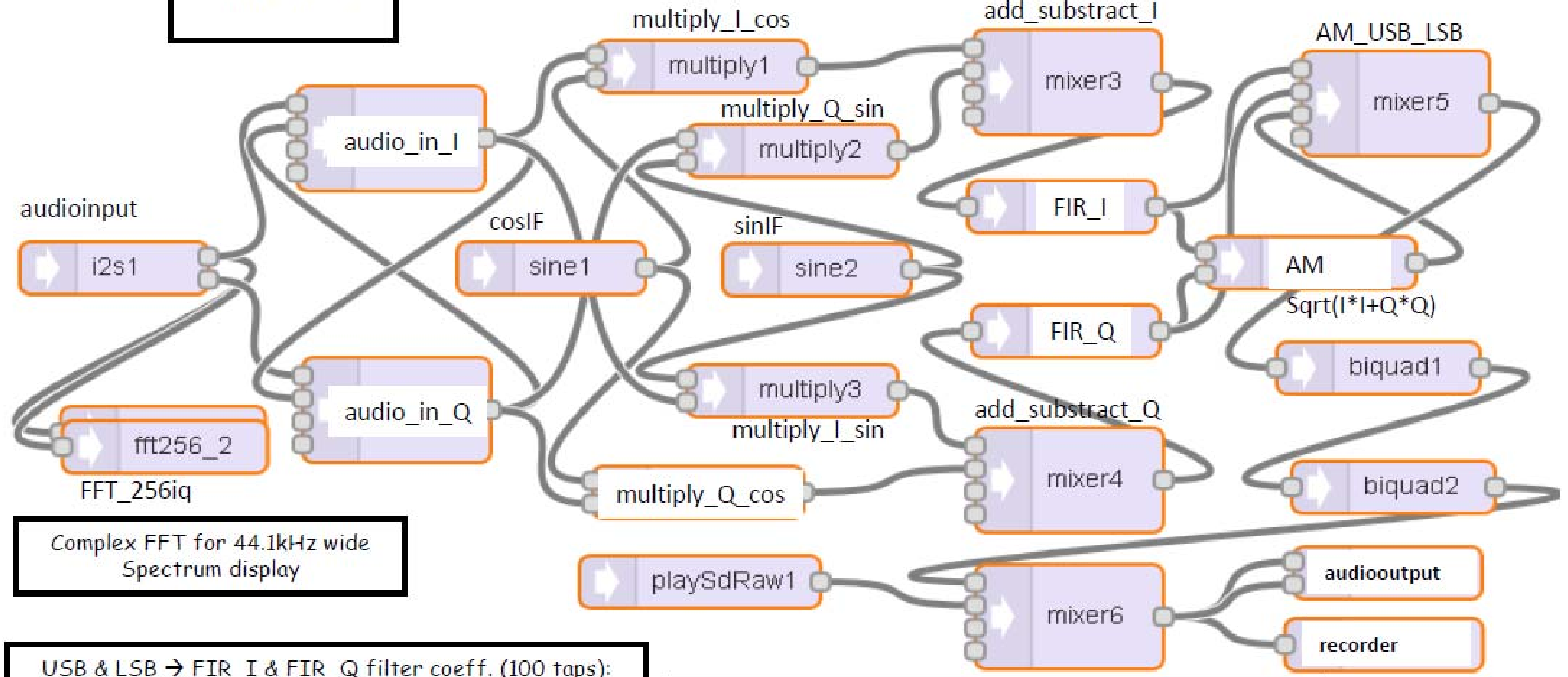
Quadrature NCOs and complex multiply: IF → baseband

Rx below or above IF: add / substract or substract/add in mixers 1 & 2

AM: FIR_I and FIR_Q: FIR LPF 66 taps

LSB or USB:
FIR_I = Hilbert-45°,
FIR_Q = Hilbert+45°
100 taps

AM_USB_LSB
gain: AM→0,0,1
LSB→1,-1,0
USB→1,1,0



Complex FFT for 44.1kHz wide Spectrum display

USB & LSB → FIR_I & FIR_Q filter coeff. (100 taps):
Hilbert 8.0kHz (+-45°), Hilbert 3.6kHz (+-45°), Hilbert 2.6kHz (+-45°), Hilbert 1.8kHz (+-45°)
AM → FIR_I & FIR_Q coeff. (66 taps):
8.0kHz, 6.5kHz, 4.4kHz, 3.6kHz, 2.6kHz, 1.8kHz
All FIR filters designed with taps with Iowa Hill FIR/Hilbert Filter designer

Biquad1 = IIR filter 8th order = 4 cascaded biquads with preset coefficients Inv. Chebychev: 1.8kHz, 2.6kHz, 3.6kHz, 4.4kHz, 6.5kHz, 8.0kHz bandwidth
Biquad2 = IIR filter 8th order = 4 cascaded biquads with continously adjustable bandwidth → Linkwitz-Riley

Recorder saves SDR radio audio as RAW audio files on SD card

## Teensy SDR Rx DD4WH software audio path – 29.11.2015

**Inspired by Rich VE3MKC, Loftur VE2LJX, Clint KA7OEI and kpc – Thanks!**

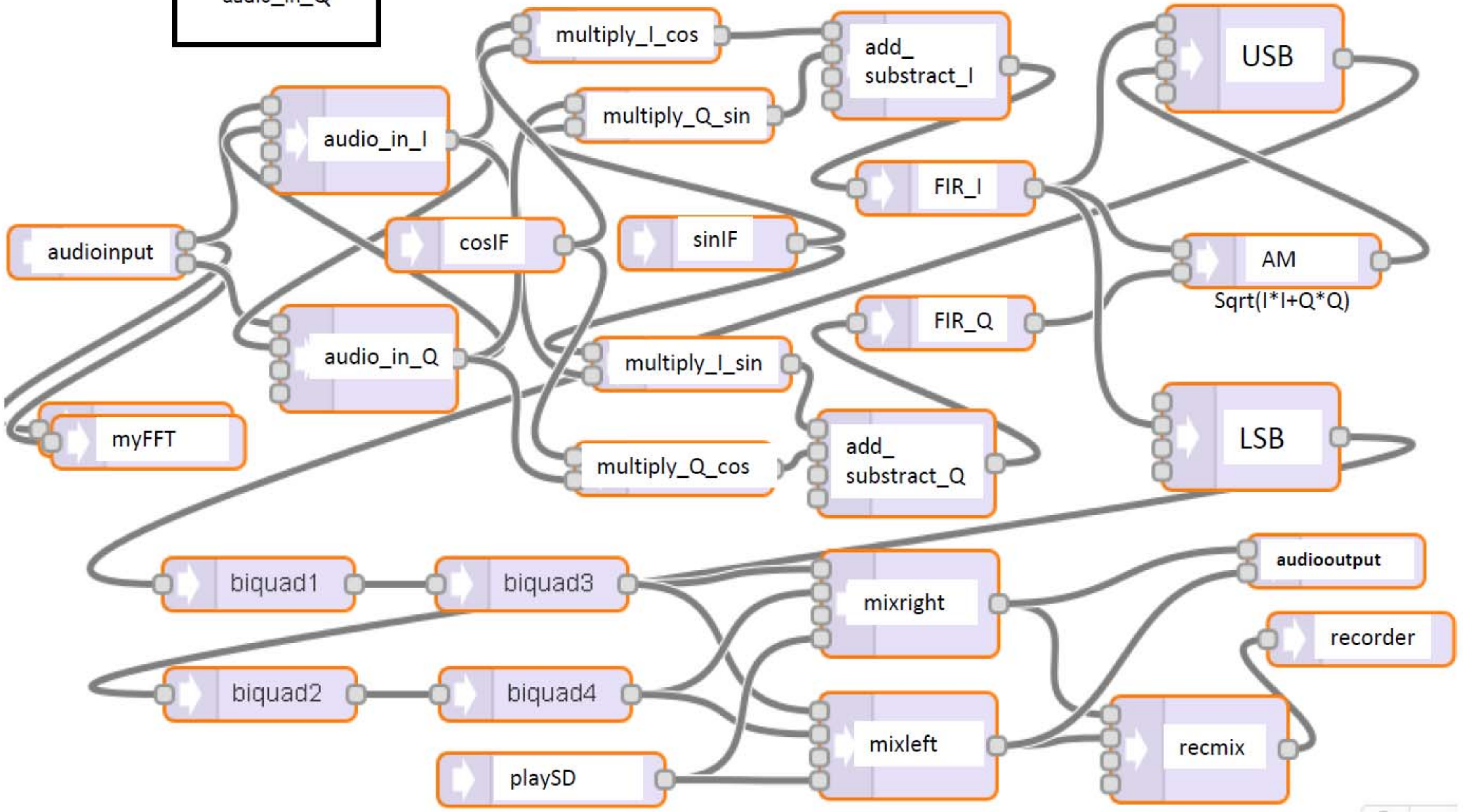
IQ Amplitude and phase calibration in audio_in_I & audio_in_Q

Quadrature NCOs and complex multiply: IF → baseband

Rx below or above IF: add / substract or substract/add in mixers 1 & 2

AM: FIR_I and FIR_Q: FIR LPF 66 taps

LSB or USB:
FIR_I = Hilbert-45°,
FIR_Q = Hilbert+45°
100 taps



This is an implementation of the Teensy SDR, the original of which was invented by Rich VE3MKC. All of the developments and new features could have never been added without his pioneering work. Thanks a lot, Rich, for this opportunity! You can find a general description of the original design in Heslip (2015).

Also check out the videos of the DD4WH Teensy SDR Receiver on youtube which explain all the functions and menu entries.

## How does this Teensy SDR radio work?

As with all IQ SDR radios, the real RF (radio frequency) input signals are transformed in hardware into complex I & Q signals (see Smith 1998 for a mathematical description of these signals). These analog I & Q signals are then digitized by an ADC (analog-digital-converter) and further processed (filtered, demodulated etc.) in software. At the end, the digital signals are again transformed into analog audio by the DAC (digital-analog-converter) and amplified in the headphone amp for listening. That´s basically all! ;-)

In general, there are four methods for a receiver to demodulate signals (Youngblood 2003): 1. The filter method, where the signal filtering occurs in crystal/mechanical filters (commonly used in superhet receivers), 2.The phasing method (normally hardware-based in old-style tube phasing receivers or in the R2Pro, see EMRFD), 3. The “Weaver” or “Third” method (nearly forgotten method and rarely implemented, but see Weaver 1956, Kainka 1983 & the excellent website by Summers 2015), 4. The “Fourth method” (van Graas 1990), which uses a quadrature sampling detector (QSD) and which has become popular due to publications and a patent by Tayloe (2001).

The Teensy SDR uses the standard approach for an IQ SDR radio (also implemented by many of the PC-based or stand-alone commercial SDR radios like the KX3, Flex-Radio series, Softrock etc.) – i.e. a **hardware QSD (fourth method)** to produce the I & Q signals and the **phasing method (second method)** implemented in software to suppress the undesired sideband in various ways.

The hardware used to produce the I & Q signals for my Teensy SDR receiver is a simple ugly-style built double balanced QSD followed by two instrumentation amps INA163 to amplify the IQ signals (based on the design by W6JL which is based on the design by Youngblood 2003). In my newest version (Dec 2015) it is a single balanced QSD frontend built on an SMD PCB by PE1KTH, thanks Joris! (Check out his genial module PCB SMD prototyping system M.T.S. LEGEO system → van Scheindelen 2015). This QSD has higher digital noise rejection due to SMD components compared to my old ugly style QSD. Local oscillator is an Si5351. The Si5351 is famous for its many spurs and high phase noise. While I can hear some spurs while tuning through the spectrum, the phase noise does not seem to limit the use of the radio, probably because the use of the Johnson counter with fLO = 4 x fRx reduces the phase noise by a factor of four. Despite the bad reputation of the Si5351, I would call this radio a high performance rig! I would love to try the Si514, which (on the data sheet) has much lower phase noise and similar low power consumption (vladn has very good experience with this LO: http://theradioboard.com/rb/viewtopic.php?f=4&t=6641). The much higher power consumption and the restricted frequency range (not lower than 4MHz, which makes MW, LW and VLF impossible) prohibits the usage of the excellent, ubiquitous and well-known Si570 as a LO in my radio.

The I & Q signals from the QSD are then input to the Teensy in the Line In input. They are converted by the ADC with a sample rate of 44.1kHz. I am not sure whether there is any built-in analog antialiasing filter in front of the ADC of the Teensy 3.1 !? If anybody knows, please drop me an Email (dd4wh.swl[ät]gmail.com).

The software path starts with the I2S audio input. From these, the signals are directly processed by myFFT, a 256point complex FFT. In contrast to the standard FFT script from the excellent Teensy Audio library (which only computes the real values), the script I use computes both the real and imaginary values of the FFT, so that they can be used to display a spectrum of the full 44.1kHz – the sampling rate (Thanks to kpc from the PJRC Teensy forum).

Because of tolerances in the hardware components, the amplitudes and phases of the I and Q signals are not perfectly matched accurately (although I used matched 33nF SMD CGO capacitors for the sampling caps). (BTW, since I switched from using poly caps as sampling caps to ceramic caps, my radio has a very high sensitivity to microphonics, this was also noted by W6JL, maybe I should use poly caps again!?).

The amplitude of I and Q can be calibrated by simply adjusting input gains in the two mixers (audio_in_I, 0) and (audio_in_Q, 0). Phase calibration is just as simple: A small amount of the I signal can be mixed into the Q channel (audio_in_Q, 1) and vice versa (audio_in_I, 1). This is equivalent to a slight phase shift of the signals (because the amplitude of a wave of different phase added to another wave determines the phase of the resulting wave). The amplitude- and phase-corrected I and Q signals then leave the two mixers and enter the complex quadrature oscillator.

Both adjustments –IQ amplitude and IQ phase- can be best done when listening to a signal of a very strong broadcast station (like Chine Radio Intl) 10kHz below (or injecting a strong RF signal with a signal generator, of course). A heterodyne signal of receive frequency fRx + 2 * IF will become visible in the spectrum display and of course hearable. The intermediate frequency is 5515Hz at the moment. In my favourite example Turkey is on fRx = 5980.021kHz, the mirror is at 5991.051kHz and can be heard as a heterodyne when you tune to 5990 in LSB/USB or DSB. Then you use the menu button to go to menu IQ Ampl. and turn the encoder until the tone is of minimal strength. Go further one step in the menu to IQ Phase and turn encoder again until tone is not hearable any more or at least very very hard to hear. I have not measured mirror rejection, but my sophisticated guess would be at least 45dB, try it and measure it by yourself, it´s amazing to hear how the mirror signal vanishes into the noise by adjusting IQ amplitude and phase in software ;-).

My SDR uses an IF frequency (currently 5515Hz = sampling frequency / 8) in order to get rid of all the noise present near DC (below 1kHz). We need to translate our I & Q signals to baseband before the real audio processing begins. This is accomplished using complex frequency translation. A complex software NCO is built and each input sample is multiplied by the real and imaginary parts of the NCO generator (for an explanation and an evaluation of its effectiveness see Lyons 2014, Wheatley 2011). The NCO uses two IF generators with 90 degrees phase shift (cosIF&sinIF) and uses four complex multipliers to multiply the I and Q signals with the NCOs. At the end, the resulting signals are added and substracted in the mixers add_substract_I and add_substract_Q. At the moment, add_substract_I SUBTRACTS the signals and add_substract_Q ADDS the signals, so that the SDR receives ABOVE the IF, if we swap ADD/SUBSTRACT in these two mixers (and the FIR_I and FIR_Q), we can receive below the IF. The user can thus choose the most convenient option (however, it is not (yet) a menu point entry, so you would have to alter the gain entries in the source code). With my current hardware setup, digital noise is lower, when Receive is above the IF. After passing the

Complex Quadrature NCOs and the complex multiply, the I and Q signals are still I and Q, but now they are in baseband and can be processed by the filters!

FIR_I and FIR_Q are the heart of the filtering process in my Teensy SDR. In AM mode, these are standard FIR filters with 66 taps. For the other modes (LSB, USB, DSB, stereoAM), these filters are phase shifted Hilbert FIR with 100 taps. In both types of filters, depending on the users choice of desired bandwidth, these filter bandwidths are choosable by passing precalculated coefficients to the filters: 1.8kHz, 2.6kHz, 3.6kHz, 4.4kHz, 6.5kHz and 8kHz. I use Hilbert filters with plus45degrees and minus45degrees phase shift. I also tried 0/90 degrees filters with the same number of taps, but not the smallest difference in sound quality was notified, so I stayed with the +45/-45 degrees filters. However, the difference in opposite sideband rejection was not systematically checked between these filters and could be very different (thanks Clint). The AM module calculates the magnitude of the filtered I and Q signals as sqrt(I*I+Q*Q). This is a very good sounding module when signals are of moderate to large strength, but if the signal-to-noise-ratio is small, this envelope demodulator sounds distorted (by its nature!) and the signal can be comparatively difficult to demodulate. In these situations, DSB and stereoAM demodulation modes can have an advantage over AM envelope demodulation. For all modes, after filtering in FIR_I and FIR_Q, the audio is passed to the mixers/demodulators USB and LSB. Strictly spoken, they do not act as mixers, but as an adder (I+Q = USB), substractor (I-Q = LSB) or switches (for AM).

**Gain adjustments for mixers UBS & LSB**

LSB mode: mixer USB (0,0,0), mixer LSB (1, -1, 0) →substraction I-Q = LSB audio in the LSB path, USB path silent

USB mode: mixer LSB (0,0,0), mixer USB (1,1,0) → addition I+Q = USB audio in the USB path, LSB path silent

AM mode: mixer LSB (0,0,0), mixer USB (0,0,1) → only AM audio switched in the USB path, LSB path silent

DSB / stereo AM: mixer LSB (1, -1, 0) → LSB in the LSB path; mixer USB (1,1,0) → USB in the USB path

Up to this point in the software path, the audio has been prefiltered by the FIR filter and demodulated according to the user defined mode (LSB, USB, AM, DSB, stereoAM). The audio is further processed in the two paths – LSB path and USB path. Each path is filtered with two cascaded IIR filters. These are very computation efficient filters implemented as four stages of biquad filters, thus each of them is equivalent to an 8th order IIR filter. The first stage in each path, biquad1 and biquad2 is implemented as an IIR filter with an inverted Chebychev response (I tried elliptic response, but those filters were not stable with fixed point calculations). The coefficients have been calculated with the Iowa Hills IIR filter designer for the same bandwidths as for the FIR filters (1.8, 2.6, 3.6, 4.4, 6.5, 8.0kHz). I also made several attempts to use other filter reponses (especially elliptic) and also modified the original audio library to be able to use more than four cascaded stages for the biquad filters. However, with six stages the IIR filters oscillated more or less heavily (resulting in very annoying noise exactly at the designed filter cutoff frequency), regardless of the desired filter responses (elliptic, Chebychev, Butterworth . . .). This is probably due to the inherent inaccuracies in the fixed point calculations of the Teensy (?), so I sticked to the originally allowed maximum number of four cascaded biquad stages. The second IIR filter stage with biquad3 and biquad4 is the standard implementation with a Linkwitz-Riley Response (two cascaded two-stage Butterworth response biquads) with 48db/octave. The inv. Chebychev filters have a more than 2.5 times steeper filter slope. Coefficients for biquad3 and biquad4 are calculated by the Teensy on the fly for the exact bandwidth, not using precalculated coefficients like the other filters.

From the biquad3 and bíquad4, the signals go to the mixright and mixleft mixers, which do exactly what their names suggest: they mix the signal according to the user-defined reception mode: LSB-only the LSB path is mixed to both ears; USB-only the USB path is mixed to both ears; AM-only the USB path (which in this mode contains the AM mode) is mixed to both ears; DSB-USB and LSB are mixed and the same mix is given to both ears; stereoAM-USB and LSB are mixed to right and left ear/channel of the stereo output; playSD-the audio output of an audio file from the SD card is mixed to both ears.

The output from the mixright and mixleft is given to the audiooutput and is converted into analog audio by the DAC. This output is mixed to mono and given to the Recorder module, which records the audio of the radio to a RAW file on the SD card (when in RECORDER mode).

## Sophisticated demodulation with passband tuning and automatic carrier detection (snap mode)

In situations when SNR is low (thus AM demodulation does not give a nice audio result), you can choose DSB demodulation mode, which often gives a much better audio resolution than AM inthese situations. However, DSB (in contrast to AM) requires the tuning frequency to be set very accurately to the carrier frequency of the signal, even more so than LSB and USB mode. If not tuned accurately, you will hear the heterodyne tone or annoying fluctuations of the signal (i.e. oscillations with the difference frequency between carrier frequency of the signal and the tune frequency). You can do two things to properly tune: do it by hand and your ear, or let the snap mode do the job!

### How does the frequency snap mode work ?

The automatic tuning process is based on determining the difference between your tuning frequency and the carrier frequency of the station you are listening to. It is based on the results of the complex FFT that is also used for the spectrum display, so the data is already there and does not consume additional computing resources of the Teensy. The values of the complex FFT are magnitude values of signal strength for 256bins across the whole spectrum of 44118Hz. That means, one calculated value for one bin holds the average magnitude for a bin bandwidth (binBW) of 172.33Hz = 44118Hz / 256. A bin bandwidth of 172.33Hz is of course not accurate enough for precision tuning, as necessary for DSB or StereoAM listening. We would need an accuracy of a few Hz to be able to listen to a signal in these modes without annoying heterodynes or oscillations of the signal. This is achievable by inter-bin three-point quadratic interpolation! You use the magnitude values of three bins adjacent to each other and use these values to interpolate a quadratic curve (which stands on its “head”) (Jacobsen & Kootsookos 2007, www.ericjacobsen.org, Thanks vladn for pointing me to this method and this and other papers!). That’s all. Now you would set up a tuning process:

I use a very simple process in my Teensy software, only two frequency steps are sufficient for very accurate tuning to the carrier:
1. determine the bin numbers, between which the maximum FFT magnitude is being searched for(bins within the passband, as I can use different USB & LSB bandwidths in DSB tuning (passband tuning) mode)
2. search for maximum FFT magnitude value in that frequency range
3. **coarse tune** to that bin (delta = deltabins * binBW)
4. interpolate with three-point-quadratic interpolater (equation (4) of Jacobsen &Kootsookos 2007) with the three bins in the centre of my frequency (in my case bins 94, 95,96 (Rx below IF)/158,159,160 (Rx above IF) as I have an IF of 5515Hz = 32 bins)
5. **fine tune**: set freq to that interpolation --> delta = binBW * (1.36 * (bin3 - bin1)) / (bin1 + bin2 + bin3); for maximum accuracy, this requires the FFT window type to be set to Hanning !
6. done!

I tested several formulas: The 2007 paper by Jacobsen &Kootsookos had the final implementation formula. As I used a complex FFT library, that already calculated magnitudes, I was looking for a formula for magnitudes. Also I wanted to use Hanning, so their equation (4) solved my problem: delta = binBW * (1.36 * (bin3 - bin1)) / (bin1 + bin2 + bin3);
This is astonishingly accurate! With a fs = 44118Hz and 256point FFT I have an accuracy of 2-3Hz with strong signals, and 4-5Hz with very noisy signals. Thats perfect with a binBW of 172.33Hz in my view.

## How does passband tuning work?

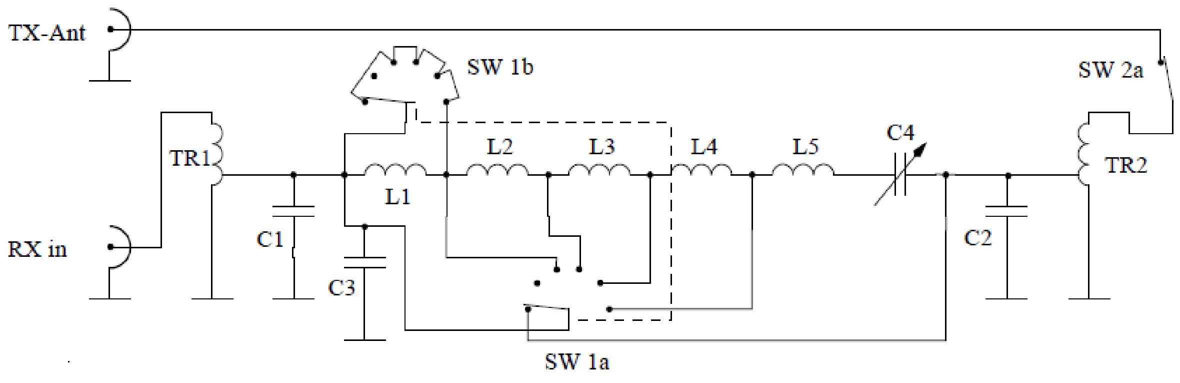
In DSB demodulation mode it is possible to use different audio bandwidths for the upper and lower sidebands. That makes it possible to prevent QRM from one sideband to enter your audio signal, but to preserve high audio frequencies fom one sideband and preserve low audio frequency response from both sidebands. Imagine an annoying sinuoid signal 1kHz in the upper sideband from the carrier. You could passband tune, so that the USB bandwidth is 800Hz, preventing the annoying QRM signal to enter your audio. LSB is set to 4kHz bandwidth, so you get a very nice high audio frequency response. Simultaneously, the low audio response is also better compared to using pure LSB demodulation, because you get the low audio frequencies (0-800Hz) from BOTH sidebands. This is what I call “passband tuning” in DSB mode. It is the mode I use most often, because it is very flexible and good sounding, especially if coupled with the snap mode, because DSB mode requires very accurate tuning. Many broadcast stations today have frequencies not accurate to the last Hz, so I use snap mode to quickly have the Teensys tune frequency snap to the accurate Rx frequency and then use the passband menu to adjust the passband to eliminate QRM.

## Details of the hardware:

The hardware consists of a Teensy 3.1 microcontroller and the Teensy audio board with the SGTL5000 codec, a 1.8´´TFT display with 160x128 pixels, a PCB (Joris van Scheindelen KTH rf design) with the QSD and LO, the preselector consisting of a variable capacitor, coils wound on a single large iron core, and a switching power supply PCB.

The signal chain for the RF input signal is the following: 1.) Preselector, 2.) Quadrature sampling detector, 3.) ADC of the Teensy audio board codec. The quadrature sampling detector uses a LO (local oscillator) signal produced by an Si5351 oscillator. The signal produced is 4 x higher than the receive frequency and is processed by a Johnson counter which divides the signal by four and this signal switches the sampling detector, which samples the signal in four capacitors. The signal is then differentially amplified by two ultra-low-noise op amps and then goes directly into the line input of the Teensy audio board.

1.) The use of a **preselection** is obligatory for an SDR that uses a QSD, because the QSD is very sensitive also to odd harmonics of the carrier signals (Youngblood 2003). I use a design very similar to the excellent and very famous BCC Preselektor (Bavarian Contest Club 2015). I added the possibility to switch in two additional capacitors in order to cover also the medium wave broadcast band. Also all inductors are tapped mono coils wound on a large toroidal Amidon T140-2 iron core avoiding saturation in front of large signals. This principle is the same as in the SCR-Preselector in order to maximise Q of the coils and the IP3 of the receiver (Heros Technology Ltd 2015). In my recent version of the Teensy SDR Receiver, I additionally use relay-switched lowpass filters, but it could also turn out, that the BCC-Preselektor does such an excellent job, that the lowpass filters are no longer necessary. However, if you do not want to use a manually tuned preselector, build your SDR with switched lowpass filters! I really like manually tuned preselectors, but your mileage may vary!



Original design of the BCC Preselektor by DL2NBU & DL6RAI. My design is a bit different – see text.

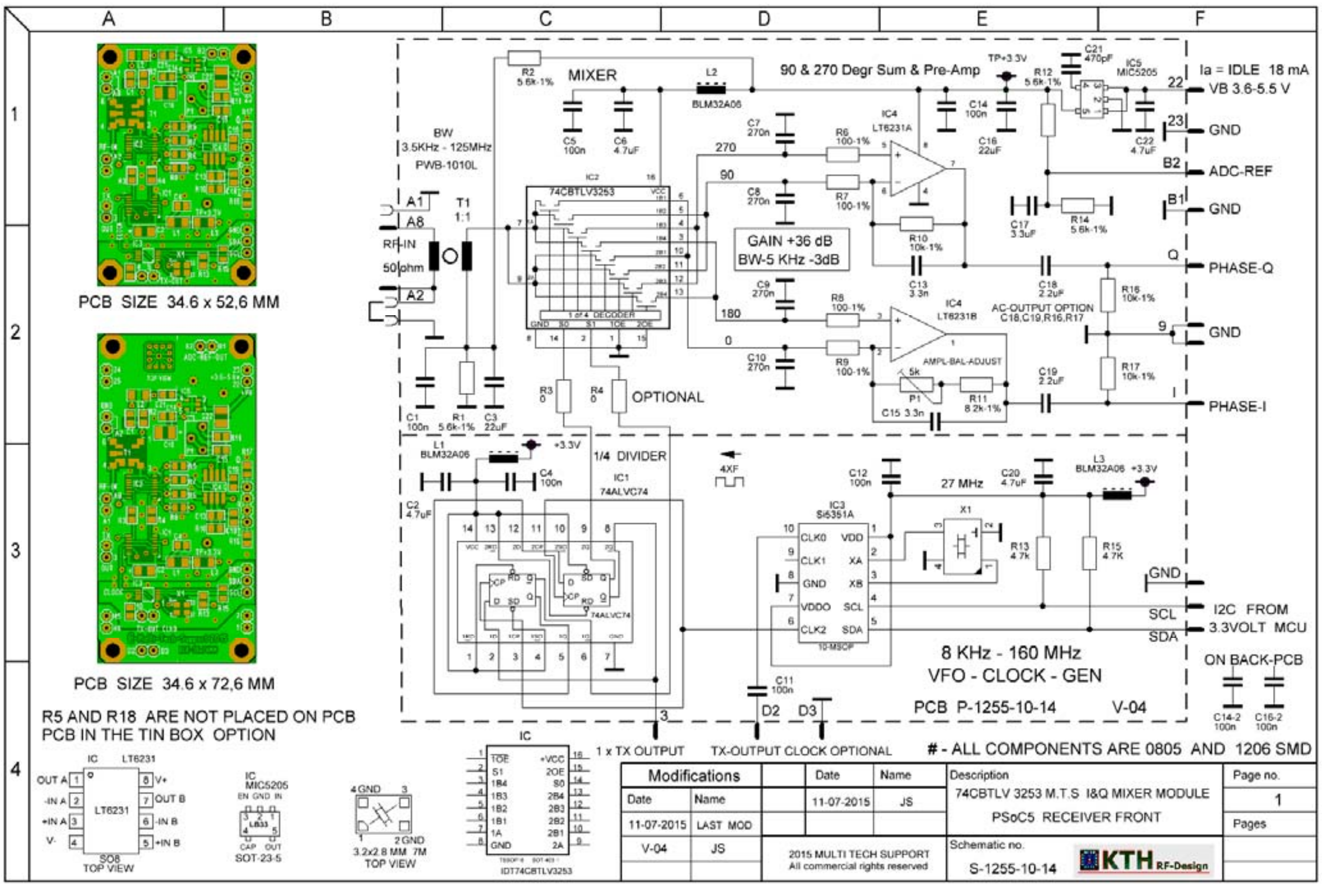
2.) I use a **single balanced quadrature sampling detector** with a 74CBTLV3253, an Si5351 as LO and a 74ALVC74 as Johnson counter and LT6231 as low noise op amps. This is all on a small SMD PCB by Joris, PE1KTH. I designed the QSD for a wider bandwidth of 44kHz (because the spectrum display is that wide). That means the sampling caps C7 to C10 were exchanged (new: 33nF) and the lowpass filtering of the op amps was also changed by changing C13 & C15 to 330pF.

Tayloe 2-sided BW = 2/2*PI*240ohms*0.033µF = 40.19kHz

I used an effective resistance of 4*50ohms + 4*10ohms = 240ohms, as the latter is the worst case of the on resistance of the 3253 switch.

Op amp LPF frequency: f = 1/2*PI*R*C = 48.2kHz with C=330pF (R = 10Kohms)

Additionally, the PCB has on it the Si5351 local oscillator which is controlled via I2C lines by the Teensy.



RF frontend by Joris KTH rf design: single balanced quadrature sampling detector, ultra low noise op amps, local oscillator Si5351 and Johnson counter (van Scheindelen [PE1KTH] 2015, PA0RWE 2015). Please contact Joris (kthkit [att] xs4all.nl) if you are interested in the design, the PCB and/or his superb LEGEO PCB module system.

**Power supplies**

My design goal was a radio with low power consumption. Therefore I tried a switching power supply, although these are famous for their production of a high amount of annoying RF. I am not experienced in switched power supplies, therefore bought a readily assembled module, the HiQPWR Power controller/Switch (http://www.technologie2000.de/hiqsdr/hiqpwr.html). This is the main power supply for a highend DDC-SDR, the HiQSDR by Helmut, DB1CC. I am very satisfied with this module, it is very usable for my Teensy SDR, the RF produced is very low and only when the module is within a few mms of the QSD PCB, a decrease in signal quality/increase in noise can be heard. However it consumes another 20mA additionally, because of LEDs and a microcontroller onboard. However, this is compensated by its high efficiency. This module produces 7.0Volts. They go into a single LDO LT1763, which produces 5.0Volts. These go to the Teensy and the audio board (which have their own 3.3V linear regulator) and the TFT display (which has its own 3.3Volt LDO regulator) after passing a two-stage RC-filter. The LT1763 also supplies the QSD PCB, which has its own on-board 3.3Volts linear low noise LDO regulator and additional chokes and caps for filtering. I think I could have lowered power consumption by using fewer linear regulators, but at the expense of higher noise level on the supply lines.

In total, the digital noise is discernable only when the radio runs without antenna and high gain and volume settings, but because I use an IF and Receive above the IF, very low digital noise is in the audio path, which is far away from 50Hz/60Hz and their harmonics and other low frequency noises. My suspicion is, that the noise from the I2C lines is higher than the noise on the power supply lines in my setup. I will have to try out bypass caps and/or RC filtering in the I2C lines (which are the communication lines between the Teensy and the Si5351 local oscillator – and of course between the Teensy and the audio board).

Power consumption is about 120mA to 150mA (depending on the band and the headphone volume)

**Conclusion**

The Teensy SDR is a perfect prototyping and experimentation platform for everyone who is interested in the functioning and setup of SDR radios. It is not a radio which you can order online and that works out of the box. However, Joris offers a PCB with Si5351 as LO and the Johnson counter and QSD on one PCB, which makes building the Teensy SDR Receiver very easy (if you are willing to solder small SMDs, which is not so difficult as you might think ;-)). You have to partially dig deep into the theory of I & Q and other DSP stuff in order to optimize the software or add new features. But, unlike other processor platforms, the audio library by Paul Stoffregen with the module approach to DSP enables us to add new features by adding small separate modules and treating problems and their solutions one by one. For me, it was (and is) a very high motivation to dig deeper into Software Defined Radio theory and try out what is possible on a rather small, cost-effective and low power stand alone processing platform – the Teensy 3.1. The setup as described above uses a maximum of 75% of the processor resources. If we used decimation/interpolation and more elegant/efficient programming (my programming knowledge is quite limited), the processor load could be significantly lower. For example, the use of a Frequency Converter module by Esteban Benito dropped the processor load by 6%. Also, the development of a new Teensy hardware with a Freescale K66 with more horsepower should make much more possible, for example features known from much more evolved SDR radio platforms (e.g. the mcHF) such as noise reduction and other sophisticated DSP functions.

**Acknowledgements**

Thanks to all contributors in forums and all SDR radio fans who helped and/or commented: Clint KA7OEI, vladn, Esteban Benito, Joris van Scheindelen PE1KTH, Pete El Supremo, Rich Heslip VE3MKC, Rob PA0RWE, and kpc.

**Useful references (some cited in the text):**

Bavarian Contest Club (2015): The BCC – Preselector. – online: http://www.bavarian-contest-club.de/projects/presel/list.htm

Hayward, W., R. Campbell & B. Larkin (2003): Experimental Methods in RF Design. – ARRL. [= EMRFD]

Heros Technology Ltd (2015): Tiny-CAT SCR Preselector. User Manual rev 01. – online: http://www.herostechnology.co.uk/pdf/Tiny_CAT_User_manual_Rev01_web.pdf

Heslip, R. (2015): The Teensy Software Defined Radio. – The QRP Quarterly July 2015: 37-40.

Kainka, B. (1983): Transceiver-Zwischenfrequenzteil nach der dritten Methode. – cq-DL 1/83: 4-9.

Lyons, R.G. (2011): Understanding Digital Processing. – Pearson, 3rd edition.

PA0RWE, R. (2015): Stand-alone SDR ontvanger met de PSoC. - Benelux QRP Club Nieuwsbrief 156: 19-21.[see also: http://pa0rwe.nl]

Smith, D. (1998): Signals, Samples and Stuff: A DSP Tutorial (Part 1). – QEX Mar/Apr 1998: 3-16.

Summers, H. (2015): Weaver article library. – online: http://www.hanssummers.com/weaver/weaverlib [2015-12-06]

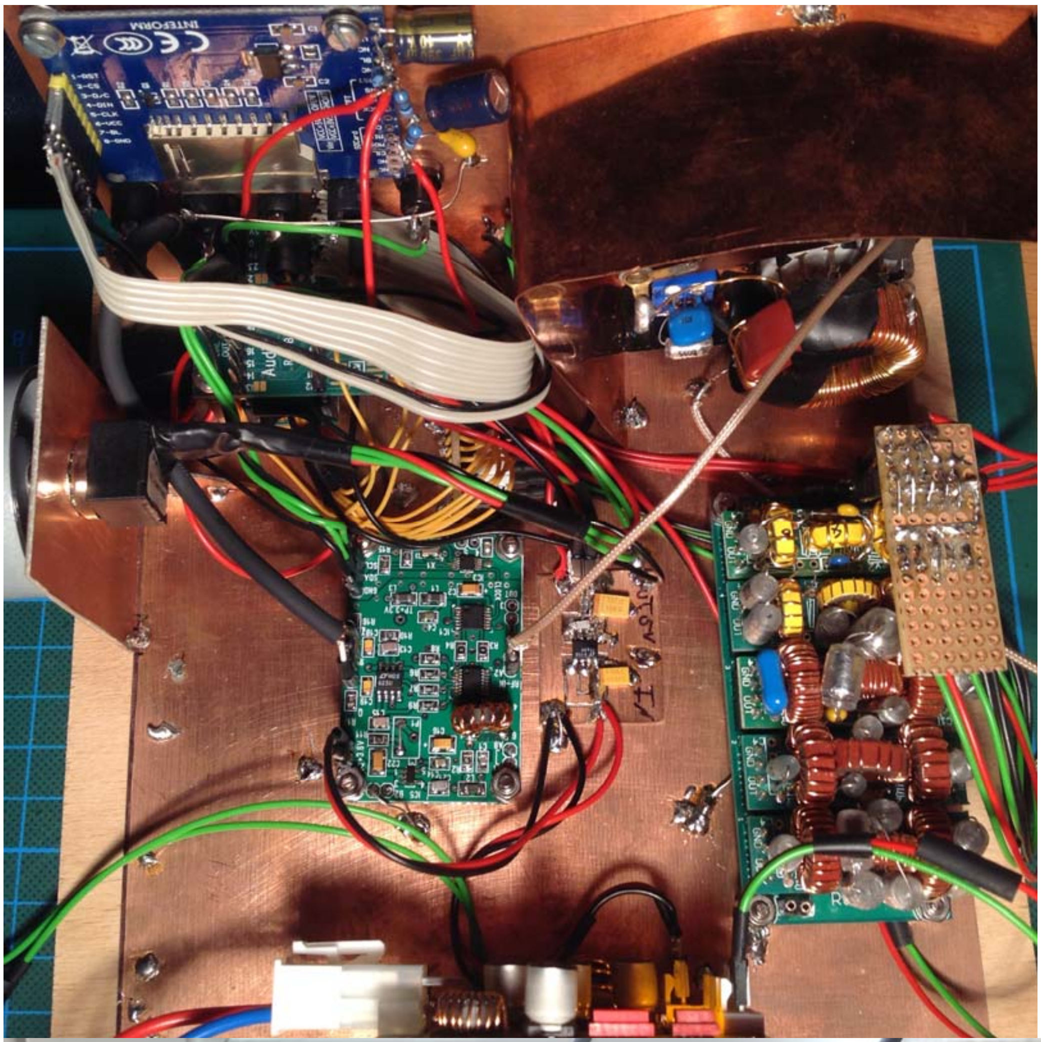
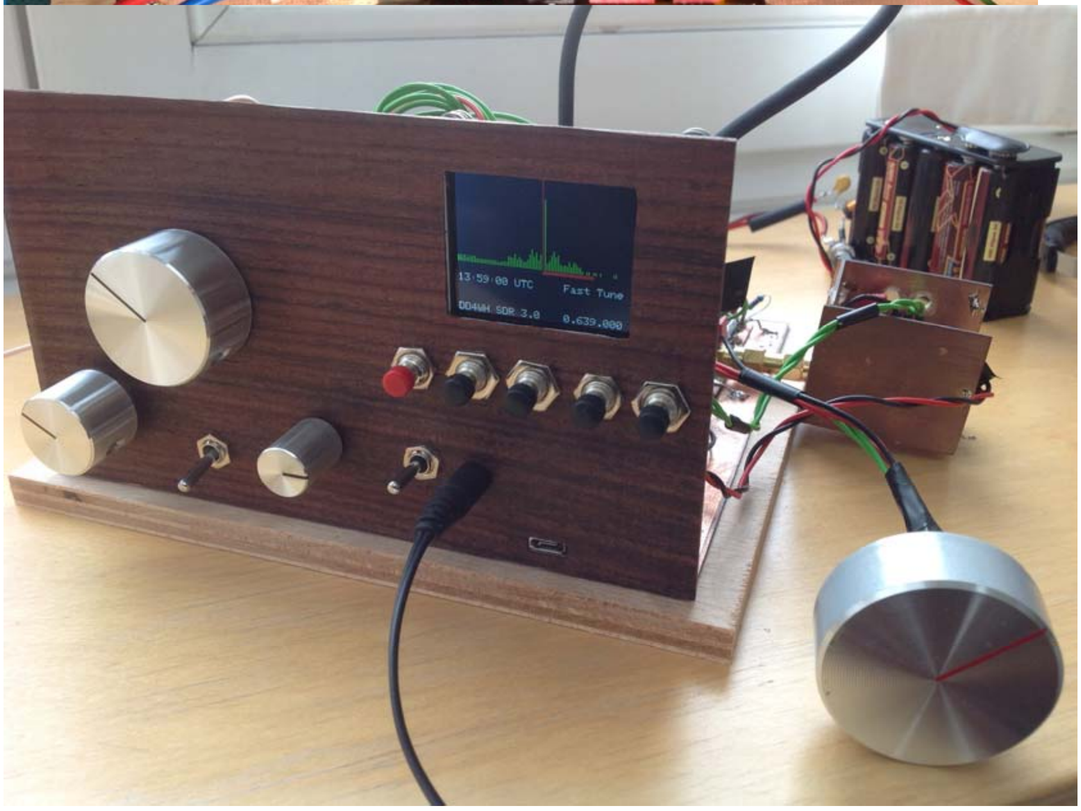
van Graas, H. (1990): The Fourth Method: Generatin gand Detecting SSB Signals. – QEX Sep 1990: 7-11.[cited by Youngblood 2003, if anybody has the original paper, I would be glad to receive a pdf]

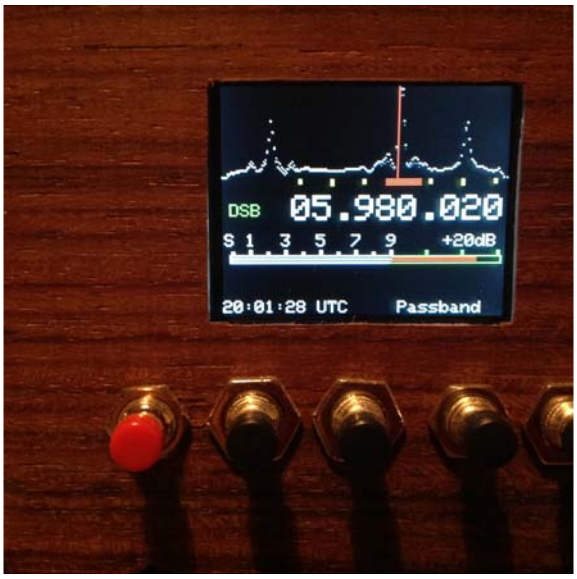
van Scheindelen, J. (PE1KTH) (2015): Electronica bouwen met M.T.S. LEGEO modulen Deel 1. - Benelux QRP Club Nieuwsbrief 156: 14-18.[see also http://pa0rwe.nl/wp-content/uploads/2015/07/sdr_schematic.gif]

Weaver, D. (1956): A Third Method of Generation of Single-Sideband Signals. – Proceedings of the IRE, Dec 1956.

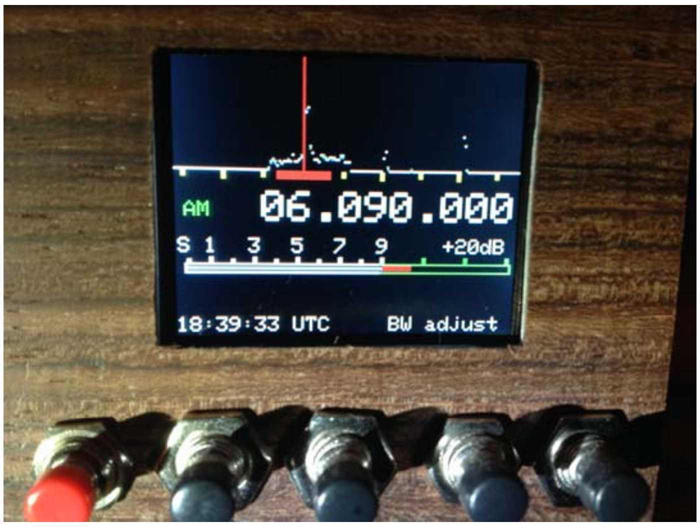
Wheatley, M. (2011): CuteSDR Technical Manual Ver. 1.01. - http://sourceforge.net/projects/cutesdr/

Youngblood, G. (2003): A software-defined radio for the masses (Part 4). – QEX Mar/Apr 2003: 20-31.

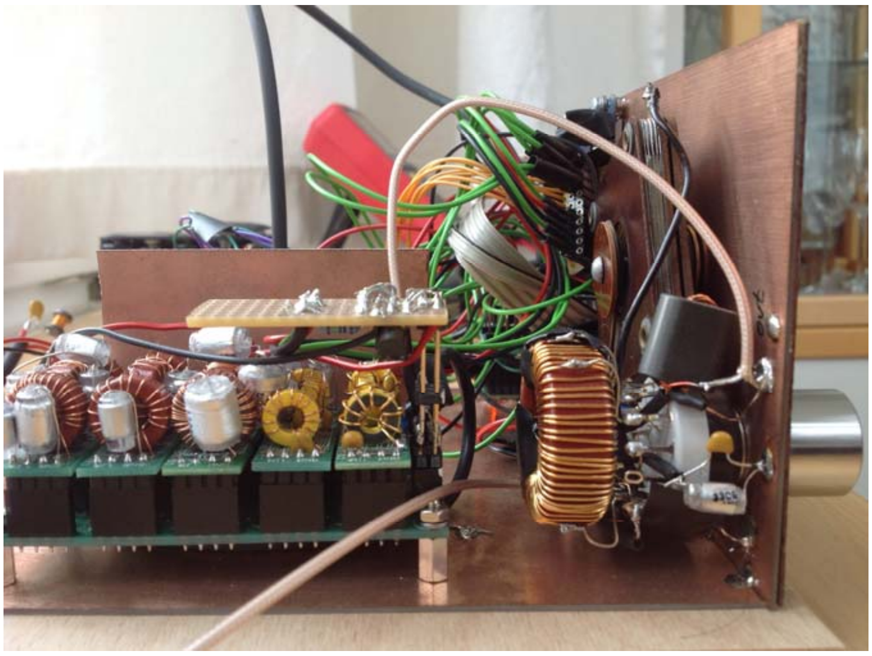


13:59:00 UTC
Fast Tune
DD4WH SDR 3.0
0.639.000

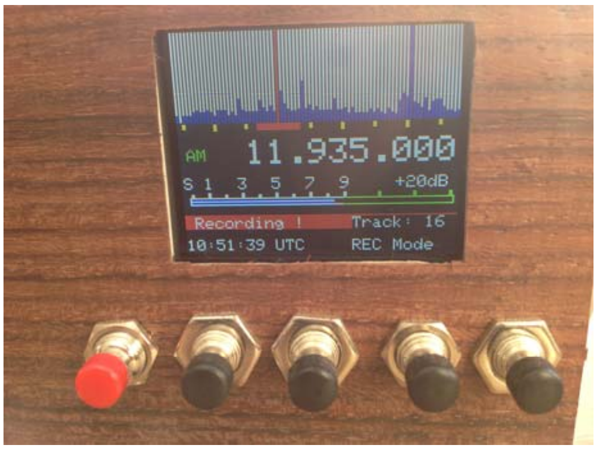


DSB
05.980.020
S 1 3 5 7 9 +20dB
20:01:28 UTC
Passband



AM
06.090.000
S 1 3 5 7 9 +20dB
18:39:33 UTC
BW adjust





AM
11.935.000
S 1 3 5 7 9 +20dB
Recording !
Track: 16
10:51:39 UTC
REC Mode

**Menus (switchable with menu switch button)**

| **Menu** | **Name ofmenu variable** | **Description** |
|---|---|---|
| Tune | TUNING | tune withencoder |
| BW adjust | BWADJUST | adjust bandwidth of all filters (FIR coefficients automatically switched, biquad1 switched, biquad2 adjusted continually) |
| Recording | RECORDING, RECORDFLAG | Recorder to capture audio from the SDR Rx to the SD card in RAW format, also in this menu ability to playback files |
| Load Settings from EEPROM | LOADFROMEEPROM | Turn encoder 64 steps to load variables from EEPROM |
| Save Settings to EEPROM | SAVETOEEPROM | Turn encoder 64 steps to save variables to EEPROM |
| Time Set (Hour & Minute) | TIMEADJUST | Turn encoder to adjust hour and minute of Real Time Clock |
| Date Set (Day &Month& Year) | DATEADJUST | Turn encoder to adjust date |
| Set Calibration Constant | CALIBRATIONCONSTANT | Turn encoder to adjust Si5351 calibration constant → temperature dependent |
| Set CalibrationFactor | CALIBRATIONFACTOR | Turn encoder to adjust calibration factor t tocalculatefrequencies |
| Set I amplitude | IQADJUST | Turn encoder for calibration of I gain (in 1/1000), Q gain stays at 1.0; gain is implemented in mixer audio_in_I channel 0 (digital gain) |
| Adjustphase | IQPHASEADJUST | Turn encoder for calibrating. Positive values mix a small amount of the I signal into Q, negative vice versa. Figures give 1/1000 of gain of I into Q or Q into I. Implemented in mixer audio_in_I channel 1 and mixer audio_in_Q channel 1 (digital gain). |
| FFT Window | FFTWINDOW | Choose FFT window: for me Hann(ing) works best, but also has the highest processor load |
| LPF Spectrum | LPFSPEC | this sets the “smoothness” of the spectrum display by applying a very basic lowpass filter to the display |

## Menus 2 (switchable with menu2 switch button 5)

| **Menu2** | **Name ofmenu variable** | **Description** |
|---|---|---|
| Tune | TUNING | tune withencoder |
| Snap ? | SNAP | turn encoder to automatically tune into the carrier of the largest station within the passband (searches for max FFT bin and then uses three-point interpolation) |
| Passband | PASSBAND | In DSB mode, encoder motion shifts passband to where you want it. Adjusts LSB and USB part separately, but simultaneously. Very good for sophisticated DXing to avoid QRM. |
| RF gain | RFGAIN | adjust analog gain before ADC |
| Bass | BASS | adjust bass level for PostProcessor (directly before DAC) |
| Treble | TREBLE | adjust treble level for PostProcessor (directly before DAC) |
| Notchfrequency | NOTCH | adjust frequency of notch filter (at the moment unused) |
| Vccvoltage check = Battery | VOLTS | shows voltage at the input of the RX |

## Switch buttons

| **Switch** | **Name ofMenu switchandvariable** | **Description (standard mode)** | **Description (in Record mode)** |
|---|---|---|---|
| Redbutton 1 | BandUPSW / band | Togglebands up | Record from radio audio on SD card: Be cautious to first adjust track number! Record function deletes the track of the defined number, if it exists |
| Button 2 | BandDOWNSW / band | Toggle bands down | play track from SD card |
| Button 3 | ModeSW / bands[band].mode | Toggle between USB/LSB and AM | leave Record mode and switch to radio mode |
| Button 4 | MenuSW / Menu | Toggle between different menus or press to exit Menu2 and return to TUNING. | previous track |
| Button 5 | Menu2 / exit Menu | Toggle between Menu2 options or press to exit menu and return to TUNING. | next track |
| Encoder push button | TuneSW / tunestep | Toggle between three different tune steps | - |

**Pin assignments**

| Pin | Name | description |
|---|---|---|
| 0 | TFT BL | backlight for display, not working at the moment |
| 1 | TFT RST 1 | display |
| 2 | TFT CS | display |
| 3 | TFT D/C | display |
| 4 | TFT DIN = MOSI | display |
| 5 | TFT CLK | display |
| 6 | AUDIO MEMCS | Audio board |
| 7 | AUDIO MOSI | Audio board |
| 8 | ENCODER BUTTON | Optical encoder Push Button |
| 9 | AUDIO BCLK | Audio board |
| 10 | AUDIO SDCS | Audio board |
| 11 | AUDIO MCLK | Audio board |
| 12 | AUDIO MISO | Audio board |
| 5V | | Vdd for Teensy, optical encoder and TFT |
| AGND | | not connected |
| 3.3V | 3.3V | |
| 23 | AUDIO LRCLK | Audio board |
| 22 | AUDIO TX | Audio board |
| 21 | Button 1 | Push button 1 = Band switch |
| 20 | Button 4 | Push button 4 = Menu switch |
| 19 | SCL I2C | I2C connection for Si5351 and Audio board |
| 18 | SDA I2C | I2C connection for Si5351 and Audio board |
| 17 | Encoder A | Encoder A = green |
| 16 | Encoder B | Encoder B = red |
| 15 | VOLUME | Pot for Volume control →analog gain of headphone amp |

|  |  |  |
|---|---|---|
|  |  | (0dB to -52dB) |
| 14 | AUDIO SCLK | Audio board |
| 13 | AUDIO RX | Audio board |
| 24 | Button 2 | Rear of Teensy → pin on breakout board. Push button 2 = Band up switch |
| 25 | Button 5 | Rear of Teensy → pin on breakout board. Push button 5 = RF gain and exit in MENU mode |
| 26 | Button 3 | Rear of Teensy → pin on breakout board. Push button 3 = Bandwidth/Mode switch |
| 27 |  | Rear of Teensy → pin on breakout board. Last pin available ! |
| 28 | Battery voltage | Rear of Teensy → pin on breakout board.<br><br>VoltCheck |
| 29 | LPF Band1 | Rear of Teensy → pin on breakout board. 1-2 MHz |
| 30 | LPF Band2 | Rear of Teensy → pin on breakout board. 2-5.4MHz |
| 31 | LPF Band3 | Rear of Teensy → pin on breakout board. < 1MHz |
| 32 | LPF Band4 | Rear of Teensy → pin on breakout board. 5.4-12.5MHz |
| 33 | LPF Band5 | Rear of Teensy → pin on breakout board. > 12.5MHz |

FIR Filters (outdated ! has to be updated)

Bandwidth, no. of taps, mode name, coefficients name, coefficient file name

1.8kHz, 100 taps, USB_24, firu24, fir_usb_24.h

2.4kHz, 100 taps, LSB_24, firl24, fir_lsb_24.h

3.2kHz, 100 taps, USB_32, firu32, fir_usb_32.h

3.2kHz, 100 taps, LSB_32, firl32, fir_lsb_32.h

4.4kHz, 100 taps, USB_44, firu44, fir_usb_44.h

4.4kHz, 100 taps, LSB_44, firl44, fir_lsb_44.h

6.5kHz, 100 taps, USB_65, firu65, fir_usb_65.h

6.5kHz, 100 taps, LSB_65, firl65, fir_lsb_65.h

ca. 20kHz, 100 taps, RX_hilbert45, RX_hilbert_45.h → this is the +45 degrees Hilbert filter

ca. 20kHz, 100 taps, RX_hilbertm45, RX_hilbert_m45.h → this is the -45 degrees Hilbert filter