SoDa – A Multimode VHF/UHF Software Defined Radio

Matthew Reilly (kb1vc)

March 20, 2014

Abstract

SoDa is a software defined radio system using the Ettus Radio USRP and its WBX VHF/UHF daughtercard. SoDa implements an all-mode (CW, USB, LSB, AM, NBFM, WBFM) radio suitable for use as an exciter/IF-strip for microwave transverters and in the VHF and UHF ham bands.

1 Introduction

In 2012 I began work on an IF radio to drive a 10GHz transverter system. The previous radio – an FT817 – had worked well, but it was time for a change. This provided an excuse to try out the USRP/N200 software defined radio system manufactured by Ettus Research. It was also the impetus to develop SoDa, the software defined radio application for the N200.

The goals of the SoDa project were modest:

- Function as a "learning lab" to explore software defined radio and digital signal processing concepts.
- Provide a practical platform for interfacing to a microwave transverter.
- Create a versatile all-mode exciter for the VHF and UHF amateur bands.
- Improve on the performance of my earlier microwave systems that used an "analog" IF radio.

Of the four goals, the first was the actual motivation. As a result, all components in the signal processing chain for the SoDa radio are developed "from scratch" though there are numerous toolkits and signal processing libraries that might have served the other goals. For instance, GNU Radio[1] offers many useful building blocks for construction of a transceiver, but I really wanted the experience of developing all the components – filters, mixers, oscillators, and the rest.

1.1 The USRP Hardware

The USRP (Universal Software Defined Radio) is primarily marketed to researchers and educators exploring cognitive radio and other emerging technologies. There are several models in the product line, including completely self-contained digital transceivers, Ethernet-connected, and USB peripheral radios. The N200 shown in Figure 1 connects to a host computer via a 1 Gbit/s Ethernet link. The N200 converts RF to and from a digital stream, while the attached computer performs most of the necessary signal processing.

Figure 1: The Ettus Research USRP N200 Software Defined Radio

The N200 is modular. The main system module contains two 100 MS/sec[1] analog to digital converters, two 400 MS/sec digital to analog converters, a field programmable gate array, and an Ethernet interface. A daughter card module implements the RF front-end. SoDa is built to use the WBX module that offers full receive and transmit operation from 50 MHz to 2.2 GHz. A simplified system diagram of the N200+WBX configuration is shown in Figure 2. Performance specifications in Table 1 are more than adequate for an IF radio.

Frequency Range	50MHz to 2.1GHz
1st IF Tuning Range (TX/RX)	+/- 20MHz
RF Sample Rate (RX)	100 MS/sec
ADC Resolution	14 bits
Noise Figure	5 dB
RF Sample Rate (TX)	400 MS/sec
DAC Resolution	16 bits
Power Out	100 mW

Table 1: System Specifications for USRP/N200 + WBX Module

[1] MS/sec: Mega samples per second, kS/sec: kilo samples per second

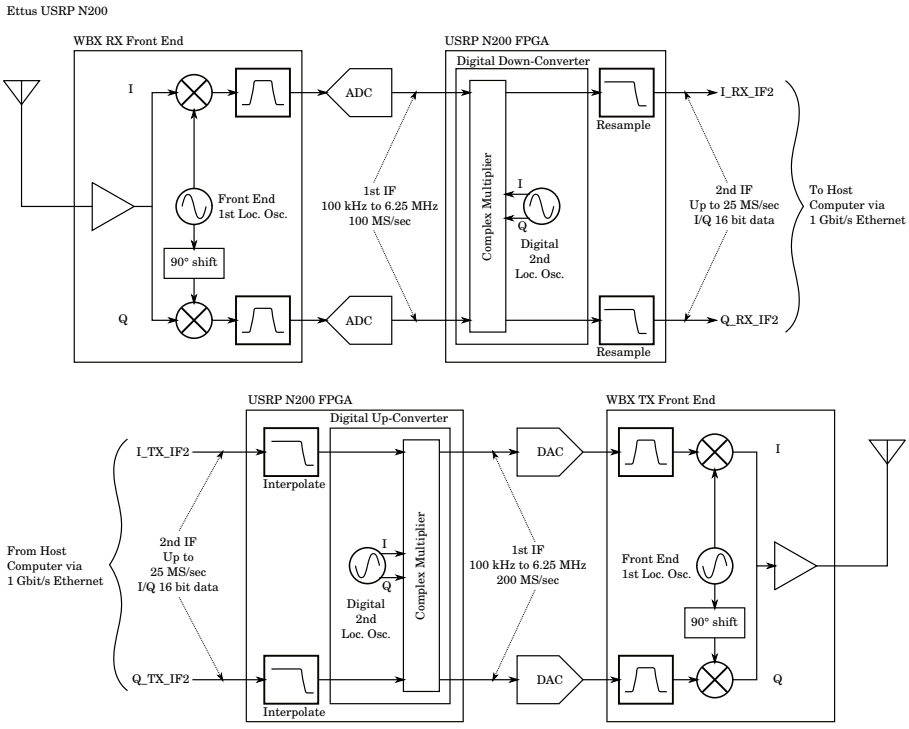

Figure 2: The USRP N200 and WBX Daughter Card

1.2 The Software

The USRP/N200 platform is controlled by software running on an attached host computer. An application controls the USRP via calls to an open-source library (`libuhd`) developed and provided by Ettus Research. The library provides functions to set oscillator frequencies on the WBX module and in the FPGA, set sample rates in the FPGA, control data transfers to and from the USRP, and manipulate control and status registers within the USRP. Figure 3 shows the major software and hardware components that make up a SoDa radio. The **SoDaServer** program handles the modulator and demodulator functions for each communication mode, software control of the USRP, and all the functions behind the knobs and buttons presented by the entirely separate **SoDaRadio** GUI program.

The **SoDaServer** program must handle a number of event streams, each with real-time constraints. The simplest programming scheme to deal with these independent tasks employs multiple threads, each tasked with a single set of related operations. Figure 4 shows the connections between the eight threads that make up **SoDaServer**.

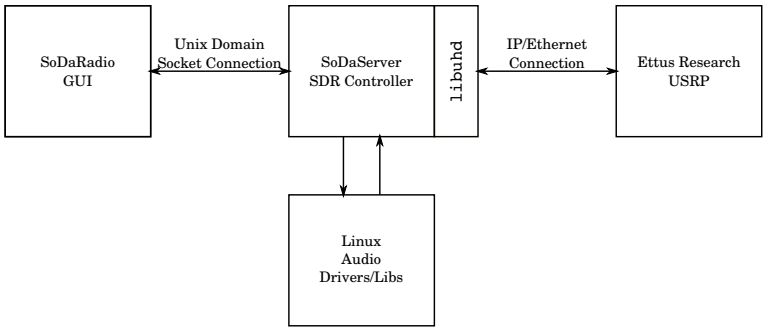

Figure 3: The SoDa Radio System

USRPCtrl handles all *control* functions between the host and the USRP. Commands that change TX or RX frequency, front end gains, amplifier settings, and transmit/receive switches all pass through the **USRPCtrl** thread.

USRPRX receives sample streams from the USRP, and downconverts the stream to the final tuned frequency. The baseband stream is passed to the **AudioRX** unit.

AudioRX resamples the baseband stream from **USRPRX** and demodulates it. The resulting audio stream is passed to the host system's audio device.

AudioTX receives audio (microphone) inputs from the host system's audio device, and creates an in-phase (I) and quadrature (Q) version of the signal that can be passed through the **USRPTX** and on to the USRP transmit path.

CWTX receives text strings from the **UI** thread for encoding into morse code. The resulting symbol stream is impressed onto a CW envelope and passed to the **USRPTX**.

USRPTX in audio modes, passes the I/Q audio stream to the USRP. In CW mode, the envelope from **CWTX** is impressed onto an I/Q low frequency carrier and passed onto the USRP transmit path.

UI receives requests from the graphical user interface and passes them onto the command bus or to the **CWTX** converter. The **UI** also passes spectrum information from **USRPRX** to the GUI.

GPS_TSIPmon manages a serial connection to a Trimble Thunderbolt GPS receiver. Collected position and time information is passed through the **UI** thread to the GUI.

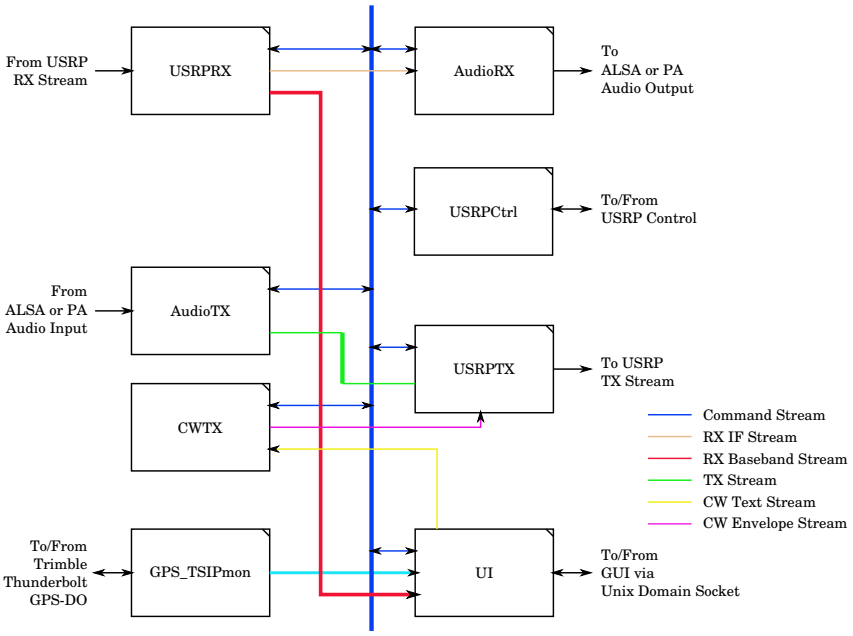

Figure 4: The SoDa Server

The **SoDaServer** threads communicate among themselves via a set of communication buses. Each bus carries messages from one source to all "listeners" on the bus. A listener subscribes to a bus, polls the bus for messages, and disposes of each message when its contents are no longer needed. Communication buses are implemented in shared memory between the threads. The communication functions are designed to limit the amount of buffer copying and other data movement. All interactions with the bus are thread-safe and protected by software locks where necessary.

1.3 Useful Background: In-phase and Quadrature Signals

Before describing the implementation of the SoDa radio, it is important to understand a fundamental concept in software defined radio design: the in-phase/quadrature representation of a signal stream.

SoDa and the USRP operate on the input signal stream as a pair of signals, called "I" and "Q". This notion of processing a signal as an I/Q pair is central to how an SDR works – almost all of the really useful DSP tricks depend on it. The I (in-phase) and Q (quadrature) signals are created in the first stage of the USRP receiver hardware when the single channel from the antenna is multiplied by the local oscillator output (to generate the I channel) and a replica of the LO output shifted in phase by 90° (to generate the Q channel). This creates what DSP engineers call an "analytic signal." If you are already familiar with

analytic signals, you may want to skip the next few paragraphs.

Consider a transmitter emitting a dead carrier at 144.285 MHz, and a second interfering transmitter at 144.284 MHz. The composite signal from the two transmitters arrives at a receiver's antenna and generates a voltage $V_I(t)$

$$V_I(t) = A\sin(\omega_d t) + B\sin(\omega_u t)$$

where A and B are the amplitudes of the carriers (on the order of microvolts, perhaps) and $\omega_d = 2\pi 144.285 \cdot 10^6$, while $\omega_u = 2\pi 144.284 \cdot 10^6$.

An analog receiver, set up in its CW_U mode, might tune a local oscillator to 134.2845 MHz, mix it with the incoming signal, filter out the lower sideband with a filter tuned to pass 10 MHz to 10.001 MHz, beat *that* resulting signal against a 10 MHz oscillator, and drive the result to a speaker. The desired signal passes through the filter and the undesired signal gets turned into heat inside the filter. This has worked for generations. It is the analog way.

But the astute reader will note that the filter required to reject the lower sideband can be quite costly, and is always something of a compromise. In the early days of SSB receivers, there were numerous schemes proposed to avoid this difficulty. DSP based receivers resurrected one of these approaches: the phasing method.[2]

For the moment, let's assume that by some extremely good fortune, our receiver was able to capture *two* sets of signals. The first set is our original $V_I(t)$ but the second set is made up of *cosine* waves like this

$$V_Q(t) = A\cos(\omega_d t) + B\cos(\omega_u t)$$

The V_I signal is the inphase signal, and V_Q is the quadrature signal.

Now let's look at what happens when we build a direct-conversion receiver by changing the first LO to 144.2845 MHz. (And let's assume that $A = B = 1$.) Now when we beat the LO against V_I and V_Q we get:

$$\begin{aligned} V_{ifI}(t) &= \sin(\omega_{lo}t)(\sin(\omega_d t) + \sin(\omega_u t)) \\ &= \tfrac{1}{2}(\cos((\omega_{lo} - \omega_d)t) - \cos((\omega_{lo} + \omega_d)t) + \cos((\omega_{lo} - \omega_u)t) - \cos((\omega_{lo} + \omega_u)t)) \\ V_{ifQ}(t) &= \sin(\omega_{lo}t)(\cos(\omega_d t) + \cos(\omega_u t)) \\ &= \tfrac{1}{2}(\cos((\omega_{lo} + \omega_d)t) + \sin((\omega_{lo} - \omega_d)t)) + \cos((\omega_{lo} + \omega_u)t) + \sin((\omega_{lo} - \omega_u)t)) \end{aligned}$$

Given that our LO and both of the incoming signals are far above 1MHz in frequency, the sum terms $\omega_{lo} + \omega_d$ and $\omega_{lo} + \omega_u$ get ignored by later processing.[3] Assuming that we can dispose of the sum terms:

$$\begin{aligned} V_{ifI}(t) &= \tfrac{1}{2}(\cos((\omega_{lo} - \omega_d)t) + \cos((\omega_{lo} - \omega_u)t)) \\ V_{ifQ}(t) &= \tfrac{1}{2}(\sin((\omega_{lo} - \omega_d)t)) + \sin((\omega_{lo} - \omega_u)t)) \end{aligned}$$

[2] This same scheme was employed to great effect in Rick Campbell's all-analog direct conversion receiver designs.[2]

[3] Actually, we can make them go away, but the details of that operation are more easily handled with complex arithmetic.

Now we play a bit of a trigonometry trick. Remembering our high school math $\sin(\theta) = -\sin(-\theta)$ and $\cos(\theta) = \cos(-\theta)$. As we've set the problem up, $\omega_{lo} - \omega_d < 0$ and $\omega_{lo} - \omega_u > 0$. So

$$\begin{aligned} V_{ifI}(t) &= \tfrac{1}{2}(\cos((\omega_d - \omega_{lo})t) + \cos((\omega_{lo} - \omega_u)t)) \\ V_{ifQ}(t) &= \tfrac{1}{2}(-\sin((\omega_d - \omega_{lo})t) + \sin((\omega_{lo} - \omega_u)t)) \end{aligned}$$

Finally, imagine that we can build a magic box that shifts any input at any frequency by exactly 90°. We'll call this box or function H and note that $H(\sin(\theta)) = \cos(\theta)$ and $H(\cos(\theta)) = -\sin(\theta)$. We'll apply this transformer to the V_{ifc} signal only, and subtract the result from V_{if}.

$$\begin{aligned} V_{ifI}(t) &= \tfrac{1}{2}(\cos((\omega_{lo} - \omega_d)t) + \cos((\omega_{lo} - \omega_u)t)) \\ H(V_{ifQ}(t)) &= \tfrac{1}{2}(-\cos((\omega_{lo} - \omega_d)t) + \cos((\omega_{lo} - \omega_u)t)) \\ V_{ifI}(t) - H(V_{ifQ}(t)) &= \cos((\omega_{lo} - \omega_d)t) \end{aligned}$$

Our resulting signal contains *only* the desired ω_d frequency![4]

The last remaining problem, is of course, that we assumed we could get both the V_I and V_Q versions of our signals. Nature would never be that cooperative. However, we can create an inphase and quadrature signal stream from a single input by adding one extra LO. Assuming an input signal $V(t) = \sin(\omega t)$ we can apply a quadrature first LO at ω_{lo} like this:

$$\begin{aligned} V_I(t) &= \sin(\omega_{lo}t)\sin(\omega t) \\ &= \tfrac{1}{2}(\cos((\omega_{lo} - \omega)t) - \cos((\omega_{lo} + \omega)t)) \\ V_Q(t) &= \cos(\omega_{lo})\sin(\omega t) \\ &= \tfrac{1}{2}(-\sin((\omega_{lo} - \omega)t) + \sin((\omega_{lo} + \omega)t)) \end{aligned}$$

And remembering that $H(\cos(\theta)) = -\sin(\theta)$, we see that V_Q is, in fact, a replica of V_I shifted by 90°.

For a more detailed explanation of the concept behind analytic signals, see [3, pp 439-464]. Lyons uses a much more powerful method for reasoning about analytic signals: complex arithmetic. "Complex arithmetic" is not nearly as scary as it sounds. Lyons' introduction is quite gentle and the illustrations are very clever.

1.4 A Few Words About Lyons

A reader of bibliographies might assume that almost everything I learned in this project came from Richard Lyons' much cited text "Understanding Digital

[4] If we had added the two signals, we would have ended up with a signal containing only the lower sideband ω_u.

Signal Processing." This is not the case, but it could have been. Rather than filling the bibliography with references to the seminal work in each area, this paper will instead refer to the relevant pages in Lyons' book wherever his text provides further support or explanation. There are many other useful, and perhaps even revered books on this topic[5] but for purposes of constructing a software defined radio, I believe that "Understanding Digital Signal Processing" is the one essential text.

2 The Receiver

The SoDa receiver chain comprises two modules. The RF component (**USRPRX**) processes the RF samples from the USRP while the AF component (**AudioRX**) filters and demodulates the received signal.

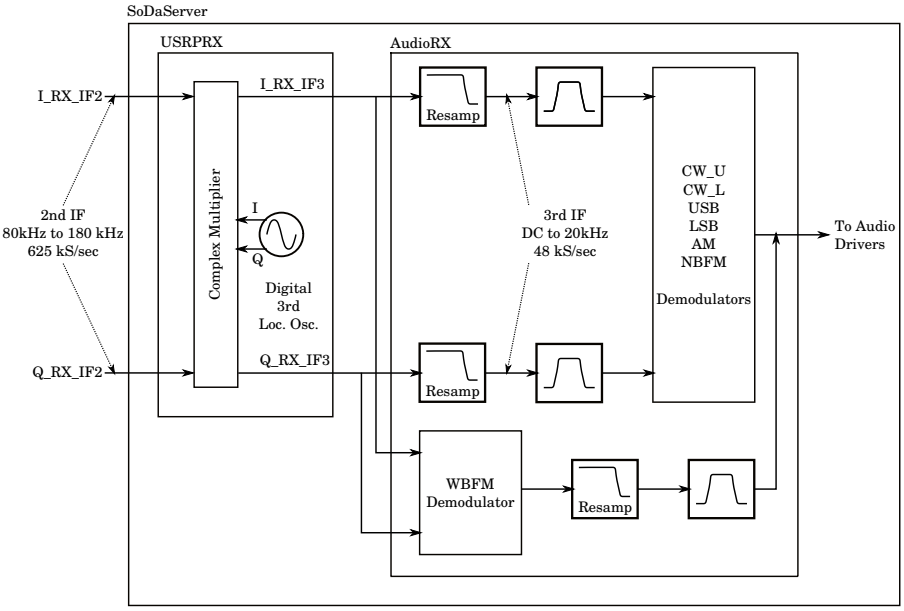

Figure 5: The SoDa Receiver Path

2.1 The USRPRX Module

The **USRPRX** module is responsible for shifting the incoming RF stream down to baseband. It maintains a quadrature oscillator that tunes from about 80 kHz to 180 kHz to cover the difference between the desired frequency and the sum

[5]Perhaps foremost among the revered texts is "Digital Signal Processing" by Oppenheim and Schafer.[4].

of the first and second LO frequencies. The first and second local oscillators are tuned so that the frequency of interest is always at least 80 kHz away from any DC output of the analog mixers, and sufficiently far away from the analog oscillators that the phase noise over the range of interest is greatly reduced from the close-in phase noise of the USRP oscillators. (See Section 4.1.2.)

The frequency shifting operation is performed with a quadrature oscillator and a complex multiplication to convert `I_RX_IF2` and `Q_RX_IF2` into `I_RX_IF3` and `Q_RX_IF3`. This is the same operation performed by the digital downconverter in the FPGA.[6] (See Figure 2.)

$$\begin{aligned} \texttt{I\_RX\_IF3} &= \texttt{I\_RX\_IF2} \cdot I_{lo3} + \texttt{Q\_RX\_IF2} \cdot Q_{lo3} \\ \texttt{Q\_RX\_IF3} &= \texttt{Q\_RX\_IF2} \cdot I_{lo3} - \texttt{I\_RX\_IF2} \cdot Q_{lo3} \end{aligned}$$

2.2 The AudioRX Module

The **AudioRX** module demodulates the RF sample stream from the **USRPRX** unit, and downsamples the stream to match the 48KHz sampling rate of the audio output device. The audio device itself is driven via the ALSA audio interface provided by the host computer's Linux operating system.[5] Figure 5 shows the organizational scheme of the **AudioRX** unit.

For most modes, the input RF stream is downsampled immediately from 625 kS/s to 48 kS/s. Next the I and Q channels are passed through a bandpass filter before the demodulation stage. Demodulation of CW_U, CW_L, USB, and LSB all follow the same path. The Q channel is passed through a hilbert transformer[3, pp 489-495] to generate a 90 deg phase shift, while the I channel is passed through an all-pass filter with exactly the same delay as the hilbert transformer. The resulting signals are added for CW_L and LSB, or subtracted for CW_U and USB to generate a single channel of audio output. Alternate sideband rejection is quite good, in excess of 50 dB, and beyond my ability to accurately measure it.

Narrowband FM demodulation uses a simple scheme that calculates the derivative of the incoming I and Q channels and normalizes these to the magnitude of the signal.[3, pp 759-760] This is a fairly inexpensive operation, though by no means "high fidelity." It is much simpler than PLL based solutions.

Wideband FM demodulation uses the same discriminator function as for narrowband FM, but the demodulation is done on the incoming RF I and Q channels before downsampling. This is done because much of the energy in a wideband FM channel would fall above the nyquist frequency for the 48 kS/s audio stream.

In both FM demodulators, the output is followed by a median filter to mitigate some of the pops and crackles that appear for weak FM inputs.

The demodulator routine produces a stream of audio buffers, each 2304 samples long. These are placed on a queue for later dispatch to the ALSA audio

[6] The reader may note a confusing change in sign for the terms involving Q_{lo3}. Downconverters must invert the sign of the Q channel in order to make the math come out correctly.[3, p 456 ff].

output. A block is removed from the queue when the audio interface is ready to accept new data.

The oscillator that generates the audio system clock is not locked to any external standard. In fact, it is likely derived from an inexpensive crystal oscillator with an accuracy far lower than 100ppm. The USRP however, is locked to a GPS disciplined oscillator. The discrepancy between the governing clocks for the two streams means that the audio system can get ahead or fall behind the USRP.

If the audio system gets ahead of the USRP, there will be a brief gap in the output audio of up to 48 mS. If the discrepancy is as much as 100 ppm, the audio stream will get one packet ahead of the USRP every 480 seconds. In practice, I've yet to see this.

For both of the host computers used so far, the audio system fell behind the USRP. This causes no great upset in the audio channel, but the delay between arrival of the signal at the antenna and arrival of the demodulated output at the speaker grew longer with time. In earlier versions of the program I noticed that the audio would fall 10 seconds behind after about 12 hours of continuous use. The **AudioRX** unit was modified to watch the queue of pending audio buffers. When the queue gets larger than 8 buffers (about 400 mS) each subsequent buffer sent to the Audio system is trimmed by 1 sample. Truncating the audio buffers would cause a regular "pop" in the speaker that might become irritating. Rather than truncate the buffer, one sample is removed from a "random" spot within each buffer. The pseudo random selection is irregular enough that the rate matching correction is undetectable to the ear.

2.3 Filters

SoDa uses digital filters in a number of places both in the receive chain and in the transmit chain. These are all built using the FFTW3[6] fast fourier transform library. The FFTW3 library is sufficiently fast that any FIR filter of more than 10 taps or so is most efficiently implemented in the frequency domain. The combination of a highly tuned FFT implementation, and much better memory access patterns with the frequency domain approach made it the logical choice. All filters used in SoDa, including the anti-aliasing filters in the resampling functions, are implemented using the Overlap-and-Save algorithm.[3, pp 716-720] SoDa's **OSFilter** class provides automatic constructors for bandpass and lowpass filters and creates the necessary save buffers and FFTW transform plans. There are four pre-defined audio filters of 100 Hz, 500 Hz, 2 kHz, and 6 kHz bandwidths. The 100 Hz and 500 Hz filters are centered near 450 Hz, while the other two filters have a lower cutoff frequency of about 250 Hz.

2.4 Sample Rate Conversion

The sample rates at each stage of the chain were selected to minimize the difficulty in conversion from one stage to the next.

Buffer sizes were chosen to make the FFT operations efficient, though they were not chosen to be powers-of-two. The FFTW3 library is quite good at mixed-radix FFTs where the size of the transform is $2^a \cdot 3^b \cdot 5^c \cdot 7^d$. This makes a buffer size of 30000 samples (for the 625 kS/sec RF stream) and 2304 samples (for the 48 kS/sec audio stream) convenient. The two buffer sizes and the two sample rates are in the ratio of $\frac{625}{48}$.

Converting from the RF input stream to the audio stream is done in four stages where each stage upsamples by a factor of 1, 3, or 4, and downsamples by a factor of 5. (See Figure 6.)

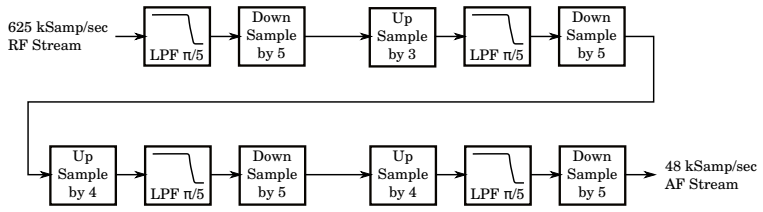

Figure 6: Resampling from RF Sample Rate to AF Sample Rate

Conversion of the transmit stream uses the same resampler code, but upsamples at each stage by a factor of 5 and downsamples by a factor of 1, 3, or 4 depending on the rate conversion stage.

3 The Transmitter

The SoDa transmitter chain comprises three modules. The RF component (**USRPTX**) forwards a modulation envelope to the USRP transmit block. The AF component (**AudioTX**) creates the I/Q channels from a single channel audio stream. The CW generation and envelope unit (**CWTX**) converts incoming streams of text into shaped pulses that modulate a low frequency carrier in the **USRPTX** unit.

3.1 The USRPTX Module

The **USRPTX** module may be the simplest unit in the SoDa radio. In CW mode, it accepts CW modulation envelopes from the **CWTX** unit and impresses them onto a 500 Hz I/Q audio tone. In other modulation modes, it simply passes the I/Q audio stream (upsampled at the RF sample rate of 625 kS/sec) to the USRP transmitter.

3.2 The AudioTX Module

The **AudioTX** module is responsible for converting incoming microphone input at 48 kS/sec to an I/Q stream, and interpolating (upsampling) to 625 kS/sec. In the case of a USB or LSB signal, the Q channel is generated by passing

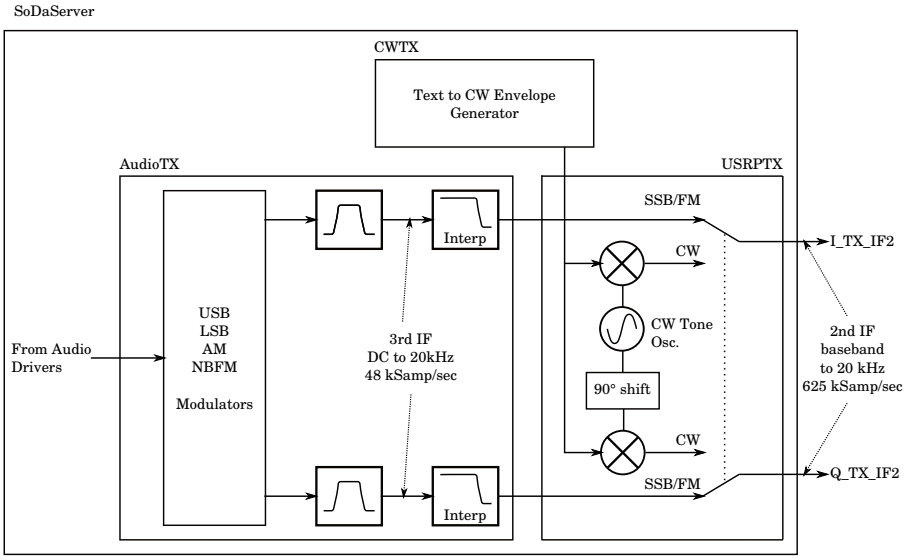

Figure 7: The SoDa Transmitter Path

the input audio through a hilbert transformer. (The I channel is created by passing the same audio signal through an allpass filter whose delay is matched to the hilbert transformer.) For LSB transmission, the sign of the Q channel is flipped. In AM mode, the Q channel is set to zero and the audio is copied to the I channel. SoDa does not yet support FM transmission modes.

3.3 The CWTX Module

The **CWTX** module converts incoming text streams (on the Control Stream, from the **UI** process) into morse code pulses. The pulses are shaped and widths are set according to the currently selected CW speed. CW speed is adjusted on request from the **UI** process.[7] The morse pulses are impressed on a low frequency "carrier" by the **USRPTX**.

3.4 Monitoring the CW Sidetone

While the CW transmitter is active, the **USRPCtrl** will tune the receiver chain to the transmit frequency. This allows the receiver sections of the radio to monitor the outgoing CW signal by listening to the leaked transmit signal inside the USRP box. While the receive chain is in "sidetone" mode, the audio gain is governed by the sidetone gain setting rather than the receive mode audio and RF gain settings.

[7]There is a knob in the GUI that causes a message to be sent to the **SoDaServer UI** thread when the CW speed is adjusted.

4 The Controller

The UHD library supplied by Ettus Research is quite complex. In general, it is thread-safe. That is, multiple threads can use the library to access a single device. But in practice, allowing multiple threads to make control changes is likely to lead to confusion, if not actual error.

The **USRPCtrl** process handles all changes to the state of the USRP that are not associated with either the receive data stream or the transmit data stream. (The **USRPRX** thread manages its own data stream, as does the **USRPTX** thread.) The controller can change RX and TX gain settings, sampling rates, transmit/receive switches, the two front-end local oscillators, and the digital downconverter and upconverter oscillators.

4.1 Tuning

Tuning with a USRP radio is spread out over three stages:

First Stage Analog LO: implemented with a very versatile synthesized oscillator, the ADF4350[7] from Analog Devices. The chip produces both I and Q outputs that span the frequency range from about 68 MHz to 2200 MHz. The chip can be used as a fractional-N or integer-N PLL synthesizer. Its tuning steps can be as small as a few hundred Hz.

Second Stage Digital Down/Up Converter: implemented as digital oscillator within the USRP's field programmable gate array. The digital oscillators can be tuned to sub-Hz resolution.

Third Stage Digital Down Converter: implemented in software as a block in the **USRPRX** process. This oscillator can also be tuned to sub-Hz resolution.

4.1.1 Transmit Tuning

Transmit tuning is very straightforward. The UHD library provides a routine that takes a target frequency as an argument and sets both the first LO (in the ADF4350) and the second LO (in the FPGA) to appropriate values. During CW_U or CW_L transmission, the requested frequency is 500 Hz below (for CW_U) or above (for CW_L) the nominal transmit frequency. For SSB or AM modes, the requested frequency is exactly the nominal transmit frequency.

4.1.2 Receive Tuning

On the receive side, the **USRPCtrl** process gives more direct guidance to the UHD library, and requests specific tuning settings for the first and second LOs. In each case, the requested receive frequency for the USRP RX chain is set to be at least 80 kHz below and no more than 180 kHz below the target receive frequency. This ensures that the DC offset component in any spectrum plots is well away from the frequency of interest. It also helps to put the analog LO tens

of kHz away from the target frequency to reduce the effects of "close in" phase noise from the PLL oscillator. The upper limit of 180 kHz keeps the third stage LO frequency well below the nyquist frequency of the 625 KS/sec I/Q signal stream.[8]

Months of experimentation with various tuning schemes showed that the fractional-N tuning supported by the supplied UHD library sacrificed a few dB in receiver sensitivity by raising the noise floor. Especially near the tuned carrier, spurs at multiples of the reference frequency divided by the fractional-N denominator (normally 1024) were quite visible in spectrum plots. Fortunately, the UHD library is open source code, so it was modified to support integer-N tuning for the first stage LO on the receiver side. (The difference between integer-N and fractional-N tuning on the transmit side is negligible.) When linked against a version of `libuhd` where integer-N mode is supported, the **USRPCtrl** process requests a first LO frequency that is a multiple of 6.25MHz and is below the target frequency. The second (digital) LO is set so that the resulting sum is between 80 kHz and 180 kHz below the target frequency.

Recent experiments have shown that results approaching those from an integer-N mode are achievable if the fractional-N settings are chosen such that the numerator of the fraction is always zero. This requires some "trial and error" tuning, as `libuhd` does not make this information directly visible.

4.2 TX/RX Switching

The USRP was designed to support full duplex communication. In particular, its transmit and receive chains have independent antenna terminals. This arrangement is convenient for driving a transverter, as I've encountered more than my share of problems with T/R switches. Running twin cables (RX labeled with a red band at each end) between the IF and the transverter is more than offset by the savings in blown-up T/R interfaces.

The WBX module contains several digital I/O pins that can be used as general-purpose I/O lines. The USRP was modified to connect one of them to a relay driver transistor that operates a keying relay. This is used to short the transverter's PTT line to ground during transmit. Putting the relay and driver transistor between the relatively expensive USRP electronics and the transverter prevents the USRP drivers from being damaged in the event of an accident (likely) or misconnected cable (certain).

The **USRPCtrl** process manages the T/R output pin and ensures that the transmit PTT line is activated well before a signal is transmitted through the TX chain, and is held on until after the **USRPTX** control process as left transmit mode.

[8] The SoDa design is quite conservative here. Because the stream is represented by both inphase and quadrature components, the practical upper limit is 500 kHz or so. Future experiments will test larger LO offsets.

5 The User Interface

The **SoDaRadio** GUI (Graphical User Interface) program and the **SoDaServer** program were built separately for several reasons.

1. The real-time requirements for a GUI are different from and less stringent than those for the USRP control.

2. Keeping the GUI and the radio parts separate allows for a cleaner and more modular design.

3. Over time, other GUI schemes might be used to allow access to the radio from handheld devices, web browsers, or other GUI instances on remote computers.

Figure 8 shows the **SoDaRadio** GUI during a contact between KB1VC and KW2T on the second weekend of the 2013 *10GHz and Up Cumulative Contest.* The waterfall diagram shows a distinct streak – KW2T's CW signal – somewhere in the range of 5 to 10 dB above the noise floor.

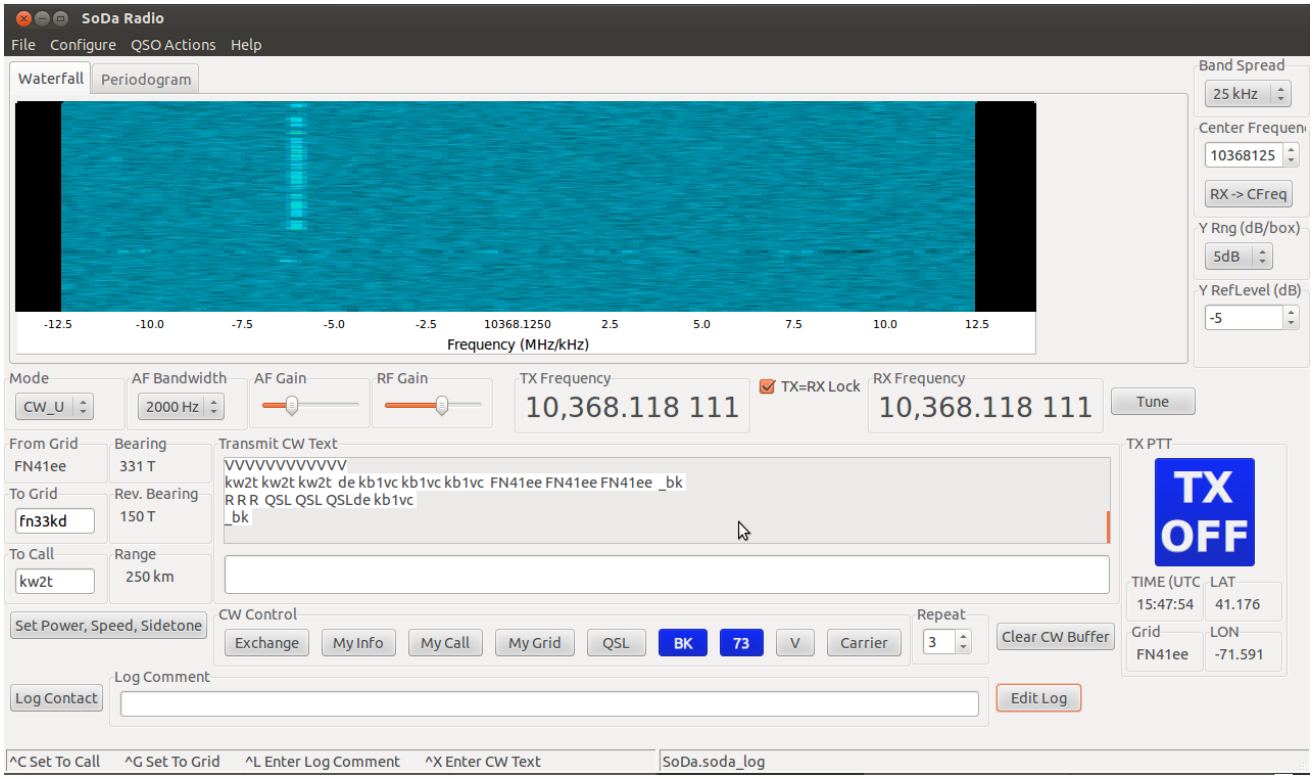

Figure 8: The **SoDaRadio** Graphical User Interface (Contact with KW2T on Mt. Equinox, VT from Block Island, RI - Sept. 22, 2013)

Experience with the waterfall display suggests that a continuous distinct trace is usually sufficient for CW copy, given a reasonable amount of patience. If the operator can *see* the signal, he can probably *copy* the signal. Conversely, I have not yet heard a signal that did not register on the waterfall display.

5.1 The GUI Implementation

The GUI was built with the `wxWidgets` cross platform GUI library, and its `wxformbuilder` GUI constructor.[8]. `wxWidgets` was chosen with the hope that SoDa might someday be ported to other platforms.

The GUI comprises five separate function groups.

Spectrum Display This is in the form of a waterfall diagram, or a *periodogram* (an X-Y plot of spectral power vs. frequency, as might be shown on a spectrum analyzer). Settings for center frequency and bandwidth are located to the right of the display. Range and reference level can be set for the periodogram, but are automatically calculated for the waterfall.

Mode, Tuning, Gain, and Bandwidth Settings These are located in the middle of the display. Tuning is normally accomplished by clicking on a point in the spectrum display. Alternatively, the **Tune** button will pop-up a tuning dialog that allows selection of the transmit and/or receive frequency. The two can be locked together with the **TX=RX Lock** button.

Navigation The `To Grid` text box combined with this station's location (the `From Grid`) produces a calculated bearing, reverse bearing, and grid-center-to-center distance in the display to aid in antenna pointing. Information received from an attached GPS receiver (UTC time and location) are shown below the big PTT switch on the right of the display. The `To Call` text box completes the information required to enter a contact into the log.

CW and PTT The white text box in the lower center of the display allows free-form text to be transmitted while in CW mode. The buttons below `CW Control` automatically generate text for the standard parts of an exchange, and can automatically trigger a transition from TX to RX mode when the exchange has been sent. The blue **TX OFF** button activates the transmitter, and changes itself to a red **TX ON** button as a reminder that the radio is transmitting.

Logging The **Log Contact** button at the bottom left of the window creates a log entry and a snapshot of the waterfall window. An optional comment may be added to the log. The **Edit Log** button pops up a dialog that allows changes or corrections to be made to the log.

Additionally, the **Set Power, Speed, Sidetone** button pops up a dialog box that allows the operator to change the transmit power, and CW speed and sidetone volume.

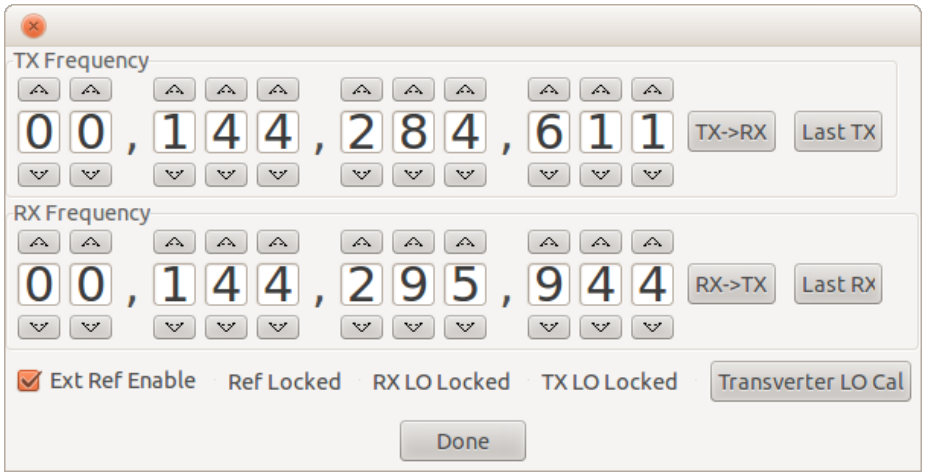

Figure 9: The Tuning Dialog Box

The tuning dialog box in Figure 9 presents two more useful features.

The **Ext Ref Enable** checkbox enables or disables the external frequency reference input to the USRP. While the local reference in the USRP is sufficiently accurate to position the VHF signal "close enough for amateur use," the external GPS reference is extremely useful for the second special feature in the tuning dialog.

The "Transverter LO Cal" function takes advantage of what would otherwise be a minor annoyance. When the USRP is used with an external microwave transverter, the transverter's own local reference oscillator leaks onto the receive RF path from the transverter back to the USRP. In the case of the Down East Microwave 10GHz transverter used here, the harmonic of the transverter LO at 1136 MHz does not greatly affect the available dynamic range of the receiver, but it is well above the noise floor. The transverter LO is not slaved to an external reference, so it can drift with time and temperature despite careful and clever engineering.[9]

While the transverter LO is not GPS disciplined, the USRP oscillators are. In "Transverter LO Cal", the USRP and SoDa **USRPRX** process are used to measure the frequency of the transverter LO using the GPS oscillator as a reference. (The **USRPRX** thread zero-beats the nominal 1136 MHz signal.) The indicated frequency is then the *actual* transverter LO frequency. In most cases it is within a few hundred Hz. If, for instance, the measurement results in a reading of 1136.000312 MHz, then the actual 10GHz LO is 9×1136.000312

[9]The simpler transverter LO design has its advantages. Because it is a simple crystal oscillator rather than a synthesized source, its spectrum is comparatively clean.

MHz or 10224.002808 MHz. The tuning in both the **USRPTX** and **USRPRX** threads is adjusted to compensate for the LO drift. This allows the SoDa radio and transverter system to operate as if all oscillators were slaved to the GPS reference, provided the operator remembers to poke the **Transverter LO Cal** button every now and then. After the transverter has warmed up for an hour or so, hourly calibrations have proved sufficient. During the warm-up phase, calibration is best invoked every few minutes.

This process is not activated automatically as it requires a second or so to get an accurate reading and the resulting audio drop-out would be quite irritating.

5.2 Communication with the SoDa Server

The GUI communicates with the **SoDaServer** via a pair of network connections using Unix Domain sockets. UDP sockets proved easier to use than IP sockets, especially for the fixed length packets that are sent over the network links. One link carries command and status packets to and from the **SoDaServer UI** process. The other link carries vectors of power-vs-frequency measurements from the **SoDaServer** UI process to the spectrum display in the GUI.

Clicking on a spot in the spectrum display, for instance, will create a tune-request message that is sent from the GUI, over the control link, to the **SoDaServer UI** process. The **UI** process then forwards the control request over the **SoDaServer** `Command Stream` where it will be seen by all of the other **SoDaServer** threads. Threads that are not "interested" in the request (such as the **AudioRX** thread) will ignore the command, while the rest will take appropriate action. The **USRPCtrl** thread will convert the tune request message into a series of calls to `libuhd`.

The **SoDaServer UI** process has no knowledge of the appearance, function, or even location of the GUI. This allows the **SoDaServer** program to be applied to other purposes, such as the construction of a spectrum analyzer, or a noise figure meter. Someday.

6 Operating Experience

SoDa has been used as the IF rig for the KB1VC efforts in the 2012 and 2013 *10GHz and Up Cumulative Contest.*

The 2012 effort exposed a number of problems with an early version of the GUI.

- Tuning was extremely awkward as the first GUI did not automatically handle CW offsets. (Clicking on a strong signal peak would tune the receiver to zero-beat.)
- The early GUI did not have a waterfall chart, and instead presented a plot of spectral density vs. frequency as one might see on an analog spectrum analyzer. Small signals tend not to deflect the trace sufficiently to allow their detection "by eye."

- The CW keyer interface was extremely awkward. Almost every CW QSO involved fumbling for the correct button and occasionally transmitting the wrong call sign.

The 2013 effort saw the introduction of a waterfall plot which helped immensely in identifying very weak signals. More importantly, it allowed me to peak the antenna on signals that were "near" the rendezvous frequency, or at least within the 25 kHz or 50 kHz window. Clicking on a stripe on the waterfall chart automatically tunes both the receiver and the transmitter, accounting for CW offset where necessary.

Between the two contests, I tried out PSK31 operation on the HF bands using my FT-817 and FLDIGI, a Linux application for digital radio. The user interface was quite well tuned for both contest operation and casual conversation, much more so than my own first attempt. While the improved SoDa UI is not as versatile as the FLDIGI interface, it proved workable in the 2013 contest.

The USRP hardware itself has held up well in portable operation, even in some unpleasantly humid (or even torrential) weather. However, the 12VDC to 115VAC inverter – required to power the the 6V DC "wall wart" that came with the USRP – generated a great deal of noise. A 12VDC to 6VDC converter was added for the second contest.

Good 12V to 6V DC to DC converters are not general stock components, but Vicor[9] makes a 5V switching regulator that is very quiet and capable of sourcing 10A. Changing the voltage sense resistive divider on the Vicor PI3302-00-EVAL1 module raised the output from 5 volts to 6 volts. Adding some input decoupling eliminated any perceptible power supply noise in the radio.

The new 12VDC to 6VDC converter was a vast improvement, but the change exposed the huge racket caused by the laptop power supply, also running off of the inverter. A 12VDC replacement power supply was even worse, creating very broadband noise that got into everything. [10] Next year will see a new supply for the laptop. In the meantime, the laptop external supply/charger is left disconnected during contacts. The laptop battery life is sufficient to allow hours of continuous operation without recharging.

The WBX module tunes a very wide range. To preserve this flexibility, it has no front-end filtering. This makes it susceptible to overload from in-band and out-of-band signals. As a transverter IF, the transverter filtering was more than adequate to suppress most interfering sources. (It could not suppress the signal from nearby VHF radios on the 2 meter liaison frequency. On the other hand, not many IF radios can.) As a VHF or UHF receiver, preselector and transmit filters are a must.

[10] Switching supplies have gotten a bad reputation, but experience with the extremely well designed Vicor units, and others, suggests that the bad reputation properly belongs to badly designed supplies, or ancient designs from the seventies. There is no need to fear switchers. The N200 employs several of them internally, as do many other modern radio designs.

7 Conclusion

The SoDa effort achieved its first goal: as a learning experience it was both challenging and rewarding. The radio performs well, and is quite good digging signals out of the noise. Over four weekends of contest operation and countless hours in the shack, it has proved itself as a transverter exciter and as an all-mode receiver.

Along the way, the SoDa project has created a set of modular building blocks for creating other USRP applications. SoDa is available on SourceForge at `http://sodaradio.sourceforge.net/` so that others might try out the Ettus USRP, a very interesting universal software radio peripheral.

References

[1] GNURadio - The Free & Open Software Radio Ecosystem. [Online]. Available: http://www.gnuradio.org

[2] R. Campbell KK7B, "High performance, single-signal direct conversion receivers," *QST*, pp. 32–40, Jan 1993.

[3] R. G. Lyons, *Understanding Digital Signal Processing*, 3rd ed. Boston, MA, USA: Prentice-Hall, 2011.

[4] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*. Prentice–Hall, 1975.

[5] Advanced linux sound architecture (alsa) project homepage. [Online]. Available: http://www.alsa-project.org/

[6] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[7] ADF4350 Wideband Synthesizer with Integrated VCO. [Online]. Available: http://www.analog/static/imported-files/data_sheets/ADF4350.pdf

[8] J. Smart, K. Hock, and S. Csomor, *Cross-Platform GUI Programming with wxWidgets*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[9] PICOR Cool-Power PI33XX-X0 8V to 36Vin Cool-Power ZVS Buck Regulator Family. [Online]. Available: http://cdn.vicorpower.com/documents/datasheets/Picor/ds_pi33xx.pdf