The Visible Radio: Process Visualization of a Software-Defined Radio

Matthew Hall, Alex Betts, Donna Cox, David Pointer, Volodymyr Kindratenko

National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign

ABSTRACT

In this case study, a data-oriented approach is used to visualize a complex digital signal processing pipeline. The pipeline implements a Frequency Modulated (FM) Software-Defined Radio (SDR). SDR is an emerging technology where portions of the radio hardware, such as filtering and modulation, are replaced by software components. We discuss how an SDR implementation is instrumented to illustrate the processes involved in FM transmission and reception. By using audio-encoded images, we illustrate the processes involved in radio, such as how filters are used to reduce noise, the nature of a carrier wave, and how frequency modulation acts on a signal. The visualization approach used in this work is very effective in demonstrating advanced topics in digital signal processing and is a useful tool for experimenting with the software radio design.

CR Categories and Subject Descriptors: I.3.8 [Computer Graphics]: Applications; I.6.6 [Simulation and Modeling]: Simulation Output Analysis; I.6.8 [Simulation and Modeling]: Types of Simulation – visual; J.2.6 [Physical Sciences and Engineering]: Mathematics and Statistics.

Additional Keywords: visualization metaphor, visualization of mathematics, radio, SDR.

1 INTRODUCTION

Radio engineers are experts at using visual tools to test and debug new radio designs. Radio frequency (RF) test equipment, such as oscilloscopes and spectrum analyzers, are used to simply and effectively display the results of electronic signal measurements. Such displays can be used in both a qualitative manner to provide an overall picture of the health of a signal, and in a quantitative one to provide detailed measurements of signal characteristics. These tools are indispensable for designing and debugging radios.

As modern radio designs evolve, however, many of the analog electronics of conventional radios are being replaced by digital hardware. In the field of digital signal processing (DSP), numerical techniques are used to perform important radio operations such as modulation and demodulation. Increasingly, the hardware used to perform the DSP is reprogrammable, so that the radio is essentially implemented in software. Such a radio is called a software-defined radio (SDR).

Unfortunately, RF test equipment cannot be used to probe and debug SDRs, as it cannot measure what is happening inside the software. For radio engineers, however, the utility of signal measurement tools cannot be overstated; tools capable of visually displaying measurements of SDR signals are therefore critically important to the design and testing of SDR's. This is where software visualization comes in play. Software visualization techniques allow us to continue to use visual information to probe and debug various processing stages in the digital processing pipeline. In addition to providing the necessary tools for radio development, new types of displays can be built that may be used to aid in the development of specific radio applications, to investigate new DSP algorithms, or to provide educational visualizations of DSP techniques.

The application presented in this paper provides an operational visualization of a complete FM SDR transceiver for the purposes of education. We target the visualization to both a general audience, and students beginning their study of digital signal processing. We would like to employ the engineer's technique of looking at spectral information as it passes between the stages of a DSP pipeline, however the engineer's tool, the spectrum analyzer, requires expertise to interpret correctly. Also, while most people have some knowledge of musical notation, the concept of mapping audio information into the visual domain does not immediately occur to them. In order to provide the audience with the necessary skills to interpret spectral data in the pipeline, we employ what we believe to be a novel didactic technique; we encode a recognizable image into the spectral information of an audio signal; when we later examine the output of a digital signal processing block, the effect of the block on the signal becomes apparent in how it transforms the image. This technique also reduces the quantitative information provided by a spectrum analyzer without eliminating it. It is still possible in our visualization to perform some useful quantitative measurements, such as counting sidebands, or finding frequency cutoffs. Therefore our visualization is not simply an intuitive introduction to SDR and DSP; it can be used to introduce students to empirical methods as well.

The paper is organized as follows. We first present an overview of software defined radio concept, and the hardware and software that we use to implement our design. We then describe how we instrument the software for visualization purposes. In order to associate the correspondence between audio signals and images, we invoke the metaphor of a player piano. We then look in detail at how each stage of our software radio acts upon a simple signal. We also take a closer look at FM modulation by varying an operational parameter and examining its effect upon a signal. We conclude by examining an interactive 3D version of our visualization, and discussing future work.

2 SDR OVERVIEW

2.1 SDR Concepts

In a conventional radio, all the signal processing functions, such as frequency translation, filtering, demodulation, etc., are implemented in analog hardware and therefore cannot be changed without altering the hardware design. While this approach has proven to be practical for a very large range of applications, there are cases in which the ability to alter radio functionality at run-time is highly desirable. Interoperability with the existing legacy systems, ability to operate with region-specific communication standards, and readiness for future communication protocols are just a few examples when a reconfigurable system is desirable.

{mahall|betts|cox|pointer|kindr}@ncsa.uiuc.edu

IEEE Visualization 2005
October 23-28, Minneapolis, MN, USA
0-7803-9462-3/05/$20.00 ©2005 IEEE.

Recent development of digital signal processing techniques and increases in available computing power have made it possible to replace rigid analog signal processing hardware with programmable digital signal processing systems that are fast enough to satisfy the needs of high-rate signal processing in modern communication systems. These developments have led to programmable and reconfigurable radios whose functionality can be changed in real-time by simply altering the software deployed on the system. SDR is characterized as "a radio that is substantially defined in software and whose physical layer behavior can be significantly altered through changes to its software" [1].

In an ideal SDR receiver, the Analog to Digital Converter (ADC) directly converts a portion of the radio frequency spectrum to digital data, which is then processed (filtered, converted to baseband, and demodulated) by a digital signal processing system and passed as the system output. Similarly, the ideal SDR transmitter consists of a DSP system that implements various signal processing functions (most notably, modulation) and a Digital to Analog Converter (DAC) that converts the DSP output to analog RF signal which is then radiated by the antenna. In reality, this approach is feasible only at very low frequency (VLF) carrier frequencies due to the performance limitations and/or prohibitive cost of high rate ADC/DAC and DSP hardware. Consequently, a more practical implementation of an SDR transceiver includes a tunable analog RF front-end and analog down/up-conversion to/from an intermediate frequency (IF) acceptable by the ADC/DAC hardware. Frequently, multiple stages of conversion and amplification occur (Figure 1) and additional filtering is needed between these stages. In such a design, the RF front-end is used to tune the radio transceiver to a particular RF band and to amplify the RF signal whereas the actual RF signal processing (modulation and demodulation in particular) takes place in the digital domain.

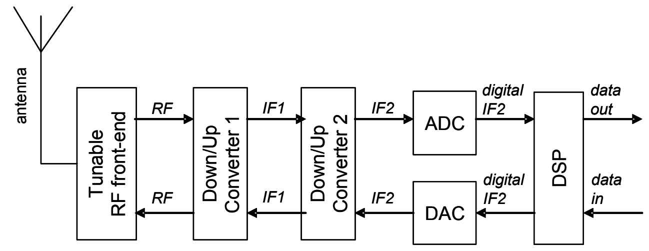

Figure 1. Practical SDR architecture.

2.2 GNU Radio

While several commercial high-grade implementations of SDR systems are available [2], the prohibitively high cost and high degree of complexity make them inaccessible and impractical for most university researchers and students who are interested in studying and experimenting with SDR technology. As a result, several simple lower-cost SDR implementations that either rely on a general-purpose computing platform or are based on an embedded DSP platform have been developed in the last few years. The GNU Radio project [3] is a particularly good example because of its user community, members of which are actively engaged in both designing new hardware front ends, and providing software interfaces to existing devices. The GNU Radio project consists of freely available software to enable the development of various radio configurations (such as an FM receiver [4]). Furthermore, the GNU Radio project is implemented using familiar programming tools, such as C++ and Python, on the Linux operating system.

GNU Radio provides a library of signal processing primitives implemented as C++ classes and the interface to link them together. Any specific radio application is built by creating a graph where the nodes are signal processing primitives and the edges represent the data flow between them. Conceptually, each primitive is designed to process an infinite stream of data flowing from its input port to its output port. In addition to a fairly complete catalog of data processing primitives, the library provides a variety of data sources and sinks which allow communication with files, networks connections, sound cards, or ADC/DAC hardware. These included components provide great flexibility for implementing software-defined radios.

The GNU Radio software library provides several primitives that correspond to the radio engineer's laboratory equipment. The most practical primitive, which corresponds to a spectrum analyzer, is the GrFFTAvgSink (Figure 2). The display shows the breakdown of a short length of signal into its constituent frequencies. From this, one can find carrier frequencies, determine the signal-to-noise ratio, or measure a number of other signal characteristics. As stated in the introduction, for an experienced RF engineer this type of display is a powerful tool.

In addition to supplying a broad range of DSP components, GNU Radio also provides a selection of working SDR applications, including a complete FM receiver [4]. The design provides a specification for the hardware front end that consists of a high-speed ADC, an RF tuner module, and an antenna. With the hardware in place, one can successfully receive and demodulate broadcast FM signals.

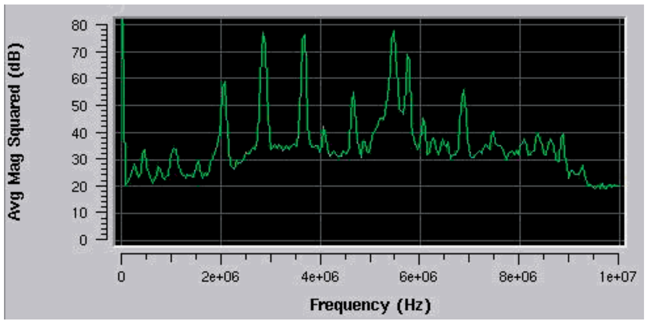

Figure 2. GNU Radio RF spectrum display.

2.3 NCSA extensions to GNU Radio

The GNU Radio software package is a capable digital signal processing library. It lacks, however, hardware for transmitting in license-free radio bands (note that the original GNU Radio hardware design [3] provides only the receiver path). Also, much of the candidate hardware for new GNU Radio applications tends to be quite expensive, discouraging potential users from experimenting with the package and constructing radios for real-world experimentation. Our recent efforts have extended the GNU Radio receiver design into a 900 MHz narrowband software defined radio transceiver [5]. The ability to transmit as well as receive makes it possible to develop and test new algorithms and protocols, to measure performance, and to provide a complete operational visualization.

We use existing GNU Radio primitives and a custom primitive to implement our FM transceiver. The structure of our SDR is shown in Figure 3. The function of the blocks will be discussed in detail in sections 3.2 and 3.3 below, however we will provide a brief overview here. The transmitter path (Figure 3) begins with the data source (VisRadioSource) which was developed specifically for this project. It encodes a grayscale image, akin to a piano roll, into an audio signal for the purposes laid out in the introduction. Briefly, it does this by taking the inverse Discrete Fourier Transform of each raster line of the image; for details, see [5]. The VisRadioSource outputs a floating point signal at a sampling rate of 48,000 samples/sec (the maximum output rate of the sound card), which is fed to the FM modulator (etgFrequencyModulator). This block was written by us because at the time, the GNU Radio project did not have a FM modulation primitive, though one is present in recent releases. The modulator generates both in-phase (I) and quadrature (Q) baseband signals, which are passed along as a complex signal. The modulated signal is mixed with a 12 kHz sinusoid (GrMixer). It is then sent to the sound card (GrAudioSink). The sound card is then connected with an audio cable to the NCSA narrowband hardware.

A complete FM receiver is a bit more complex (Figure 3). The NCSA narrowband hardware is connected to the sound card. The signal is read by the sound card (GrAudioSource) and fed into an automatic gain control stage (GrAGC). The signal is then passed to a 12 kHz. mixer (GrMixer) operating as a down-converter. The signal passes through a low-pass filter (GrFIRfilterCCC) followed by the FM demodulator (VrQuadratureDemod) followed by a low-pass audio filter (GrFIRfilterFFF). Typically, it is then sent to a sound card (GrAudioSink) where it can be listened to.

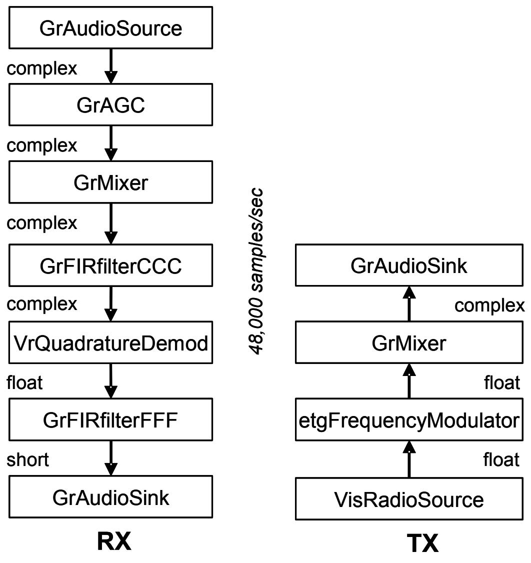

Figure 3. NCSA GNU Radio FM transceiver block diagram.

3 SDR Visualization

3.1 Visualization metaphor

The block diagrams of the FM transmitter and receiver and the frequency-magnitude plot introduced in the previous section provide a useful description of how the radios work to someone familiar with the radio engineering, but provide little insight to the radio novice. The diagrams reveal neither how each block functions, nor how the data passing through each block is transformed. Ideally, one would like to see what happens to the signal at each processing stage, how its spectral characteristics change, how much bandwidth it occupies, how sampling rate affects its bandwidth, how each block's internal parameters affect it, etc. The frequency versus magnitude plot (Figure 2) is an excellent tool for the job, however due to the amount of quantitative information it provides, and the rapidity with which it changes, it requires training to properly interpret. Instead, we use a variant of the frequency-magnitude plot called the spectrogram, which provides a less complicated view of a signal's spectrum, and also preserves the signal history. To motivate the use of a spectrogram to the viewer, we introduce it by employing the metaphor of an old-fashioned player piano.

In a player piano, a roll of paper (known as a piano roll) is fed through the player piano mechanism, where tiny perforations in the paper use a simple mapping to control the notes played by the piano. Holes on the left of the roll correspond to low notes and holes on the right correspond to higher notes. The speed with which the roll is fed through the mechanism determines the music's tempo. In essence, a piano roll is much like a spectrogram in which the presence or absence of a hole in a certain column denotes the presence or absence of a specific frequency – i.e. it shows a binary amplitude. The player piano analogy is not quite sufficient for our purposes; we need to be able to represent many more frequencies than a piano, and must also represent intermediate amplitudes as shades of gray. The similarities, however, are adequate to provide the viewer with an intuitive metaphor.

To visualize data as it flows through the radio, we examine the spectrum of the signal after it passes through each functional block; in essence, we treat each block as if it reads in a piano roll, performs an operation on it, and outputs another roll, which is then fed to the next block in the pipeline. To provide this representation, we instrument the transmitter and receiver code with visualization taps, which can connect to the output of each functional block. These taps do not affect radio function; they gather samples from their input, periodically create a line of a spectrogram from the gathered data, and then send the spectrogram via a TCP connection to a client. In our visualization, the taps output a spectrogram approximately every 1/24th of a second (corresponding to 2048 samples acquired at the sampling frequency of 48 kHz). The visualization tap includes a prescaler, and outputs a line of byte-sized data. A client does not need to process the data further to assemble each line into a grayscale image of a piano roll. Throughout this section, we will use a simple client to visualize the data. At the end of the section, we will look at a more advanced client that embeds the data inside a functional block diagram.

We also have the ability to add an audio tap at any point in the pipeline, which we do to sonify the output of a particular processing block.

3.2 Visualizing the FM transmitter

We begin by examining the radio as we transmit and receive a simple tune. The image encoder that we describe in section 2.3 acts like a software version of the player piano for our visualization. We can feed our transmitter with an image similar to a piano roll; see Figure 4a. Our software player uses sine waves to play the tune, and plays the image at a rate of approximately 12 rows per second. If we listen to the signal at this point, we hear a rendition of Mary Had a Little Lamb, followed by a C Major chord. We should point out that in order for signal features to be visible in print, we have thickened the lines of our input image. When viewing the full size visualization, we use only a single column per pitch.

The signal passes from the player through the FM modulator (Figure 4b). We see that for the single notes, the modulator produces a pattern of overtones. These patterns are fully described by a family of mathematical functions called the Bessel functions. For the chord, the modulated signal is much more complex; it is obviously more than the sum of its component notes. The important thing to note, however, is that frequency modulation spreads the frequency information across the spectrum. This redundant information will be helpful when we reconstruct a noisy signal, but is also requires greater bandwidth than does the unmodulated signal. In the framework of our player piano metaphor, this means that we need many more keys on the piano to represent the modulated signal than to represent the original. Our images, therefore, must be very wide. Listening to the signal at this point, the notes in the tune are recognizable, but distorted by their overtones. The modulated chord, however, sounds quite unlike its unmodulated counterpart.

After frequency modulation, the signal is mixed with a 12 kHz carrier wave that simply shifts the signal to a higher frequency, or in our visualization, shifts the signal to the right (Figure 4c). Note that this shift does not widen the spectrum. Typically, this step is done to shift the base signal, such as an audio signal that spans a frequency range of 0 to 48 kHz, up to a radio frequency that can be efficiently transmitted by an antenna (often in the megahertz or gigahertz range). In fact, the NCSA narrowband hardware used in this project includes a hardware 900 MHz mixer that does exactly that. The reason we also perform this relatively small shift in software is not just for illustration. Commodity sound cards for PCs typically do not output frequencies below 10 Hertz, since 10 Hertz is below the threshold of human hearing. After a signal is modulated, however, important information is carried at these lower frequencies. If they are suppressed by the hardware, the original signal can not be accurately recreated. Thus, we need to move the center of the modulated signal away from this "frequency hole". The frequency on which the signal is now centered is called the carrier frequency; we chose 12 kHz as our carrier frequency because it is easy to implement a fast and accurate mixer at 1/4th the sampling frequency, but we could have used an arbitrary value.

Those unfamiliar with communication theory may be puzzled by the high and low frequency symmetry exhibited in Figure 4. It is beyond the scope of this paper to explain why this symmetry occurs, for this we refer readers to [9]. However, in short, these "mirror frequencies" are a result of the discrete nature of the sampling process. It is also worth noting that in Figure 4c, the high frequency components that are shifted above 48kHz wrap around into the low frequency portion of the spectrum. Without going into too much detail, the mathematics of DSP dictate that any sampled signal can only represent frequencies less than or equal to the Nyquist frequency, defined to be half the sampling rate [9]. If a signal contains any frequency components higher than the Nyquist frequency, an effect called aliasing occurs, in which those high frequency components are mapped back below the Nyquist frequency. In the case of our FM transmitter, we use an effective sampling rate of 96 kHz; thus the Nyquist frequency is 48kHz. Consequently, when we shift the signal up to the 12 kHz carrier frequency, those portions of the signal above 36 kHz which were shifted above 48 kHz are mapped back down into the 0-12 kHz range. This counterintuitive behavior does not occur in the continuous realm of analog radio; it is strictly a result of the mathematics of discretely sampled signals.

Next, the signal is sent through the sound card to the NCSA narrowband hardware. One subtlety of the modulation process not shown in the visualization is that our modulator produces a complex signal; that is, its output is of the form x+iy. It is not necessary to produce a complex signal. One can design an FM modulator to produce only real valued output (which would be simply the real valued component of the complex signal). We produce the complex output, however, because by assigning the real component of the signal to the left speaker channel and the imaginary component to the right, we can use the full capacity of the sound card to provide more bandwidth than with a single channel. In our analogy with a player piano, having the bandwidth of both channels available allows us to transmit a wider range of signals in the same way that having 88 piano keys instead of 44 allows us to play a wider range of music. Before we can transmit, however, it is necessary to combine the two channels into a single analog signal; for this, the narrowband hardware uses a technique called quadrature mixing ([6],[7]). At the same time, it shifts the signal up by 10.7 MHz. A second hardware stage mixes the signal with a 900 MHz carrier wave (the same operation that we performed earlier in software) and passes the signal to the antenna so that it may be transmitted. This quadrature mixing technique is not strictly necessary; if we didn't want the extra bandwidth we could, in fact, send only the real component of the demodulated signal, mixed with a 12 kHz carrier, to the sound card. The analog hardware could then (theoretically, at least) prepare this signal for broadcast in a single step by mixing it with a radio frequency carrier.

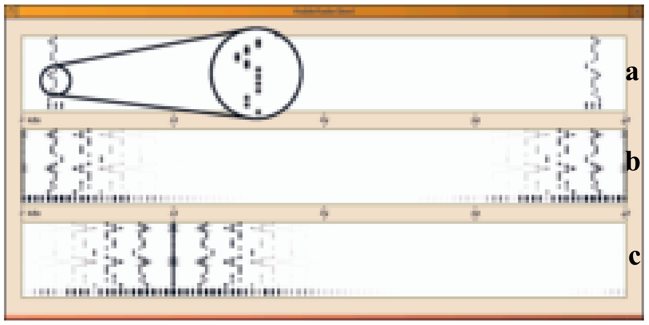

Figure 4. The FM transmitter signal processing stages.

3.3 Visualizing the FM receiver

The FM receiver operates much like the transmitter in reverse, but it includes extra components to deal with signal decay and noise. Figure 5 shows different stages in the signal-processing pipeline of the receiver. In part a, (VisRadioSource-to-GrAGC path) we see the signal as we receive it from the sound card. It is immediately apparent that although the input signal (Figure 5a) bears a resemblance to the transmitted signal, it is much weaker. In truth, this weakening can be attributed to the improper match between the input gain on the sound card and the input from the NCSA narrowband hardware, however this artifactual signal decay is used to illustrate a very real phenomenon: radio signals weaken in proportion to the inverse square of the distance from the transmitter. The signal received by a radio 10 miles from a radio station, therefore, is hundreds of thousands of times fainter than that received by a radio 100 feet from a radio station. The received signal therefore requires significant amplification, but such amplification (or gain) cannot be performed using a constant factor because the strength of the input signal can vary by orders of magnitude. We must amplify the signal just enough to place it within a known range – this process is known as Automatic Gain Control (AGC). Generally, one must calculate the average power of the input signal over a short period of time, and then use that to determine the gain. FM signals have constant amplitude, however, because all of the information they carry is contained in the frequency makeup of the signal. AGC can be very simply implemented on a complex, frequency modulated signal by using DSP techniques. By dividing each sample by its norm, we obtain a signal with a magnitude of one. While the NCSA narrowband hardware has a built-in AGC stage, we also apply the software defined AGC to the incoming signal. Figure 5b shows the output of this operation. We then shift the signal down in frequency by mixing it with another carrier wave. This operation is almost identical to the one we performed in the transmitter; the difference is that we mix with a time-reversed 12 kHz wave. From a visual perspective, this simply shifts the image to the left (Figure 5c).

After the signal has passed through the AGC and carrier removal stages, it is now evident that the signal contains a significant amount of noise. Noise in an input signal comes from a variety of sources such as imperfections in the transmit/receive hardware, interference from other radio sources, and from natural background radiation. It is not an exaggeration to say that the majority of effort in designing a radio is in reducing noise from a signal. This is not a simple problem; there is no way to automatically decide whether part of a signal is information or noise. From a visual perspective, we might ask if a gray pixel represents a note being played or a lightning strike miles away.

Although there is no easy answer to this question, we can use our knowledge about the type of signal being sent to reduce noise. For instance, even simple visual inspection of our transmission pipeline reveals that almost all of the non-zero spectral information in the modulated signal lies between zero and 12 kHz. We can therefore reasonably infer that any component of the signal above 12 kHz is almost certainly noise. Another factor to consider is that there appears to be an artifact clearly visible at 12 kHz in Figure 5c, perhaps due to the sound card hardware. We can rid our signal of both a substantial amount of noise and the artifact merely by zeroing out all of the frequency information from 11.5 kHz on up in an operation called filtering. We wish to remove all portions of the signal above a certain frequency. We therefore employ a type of filter called a low-pass filter, which passes frequency information below a certain level called the cut-off frequency. As one can see in Figure 5d, low-pass filtering is a brute force technique because it is insensitive to whether or not the filtered high-frequency information is actually noise. It also leaves unchanged noise in the lower part of the spectrum. The substantial proportion of the noise eliminated by low-pass filtering reveals it to be a powerful technique, however. For this reason, low-pass filtering and others of its kind (high-pass, band-pass, and band-reject filtering) are the tools most widely employed to eliminate noise.

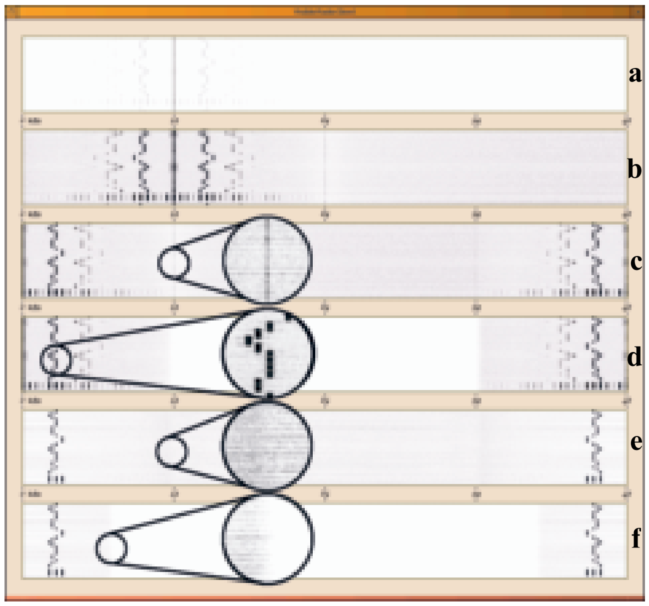

Figure 5. Signal processing stages in the FM receiver, with details of a sound-card artifact (c), DFT leakage (d), noise reintroduced by modulation (e), and low-pass filter cutoff (d).

The detail for Figure 5d exhibits a slight horizontal smearing between successive notes (see inset in Figure 4a for comparison). This artifact is purely visual, and is not audible in the sonification. It occurs due to the fact that the visualization is formed by taking the spectrum of the signal by taking the DFT of about 1/12th of a second of signal, in essence trying to recreate the piano roll from the signal. As the transmitter and receiver are not synchronized, it is likely that when a note changes pitch that it does so in the middle of the 1/12th of a second interval. Due to the mechanics of the DFT (this issue is known as DFT leakage [9]), this appears as a smear rather than as two distinct notes. Adding some form of synchronization would be possible by modifying the transmitted signal, but would complicate the radio design. We decided to keep the transmitter and receiver simple, and accept the artifact.

After filtering, we demodulate the signal. The result of this process is shown in Figure 5e. At this point in the receiver, our original signal is clearly recognizable, but some noise is still present. In fact, demodulation has introduced some noise into a previously filtered portion of the spectrum. If we sonify this stage of the radio pipeline, this noise is clearly audible as hissing. Again, we employ knowledge of the original signal to help us reduce the noise. Since the original signal contained only frequencies below 6 kHz, we pass the signal through a low-pass filter as before. This time, however, we set the cut-off frequency to only 6 kHz. The final result still contains some visible noise, and we can see the smearing artifact (Figure 5f). Aurally, however, we have quite faithfully reproduced the input signal.

In reference to our claim that our visualization allows quantitative techniques to be used, notice how easy it is to read the low-pass cut-off values from Figure 5 (d and f). Our calibration bar is a bit coarse, but it is clear that the signal in Figure 5f falls off sharply just below 6 kHz.

3.4 Using visualization to understand FM

Now that we have successfully transmitted and received a simple tune, we can return to examine the frequency modulator. While the theory behind FM is beyond the scope of this work, a brief look at the real form of the equation for frequency modulation is in order [8]:

$$x(t) = \cos(k\int_0^t u(s)ds)$$

Here, u is the input signal, x is the modulated signal, and k is a unitless parameter called the modulation index. The modulation index controls how much bandwidth is used by the modulated signal, and can be estimated by using a formula known as Carson's Rule [8]. To visualize the effect that changing the modulation index has upon the signal, we again transmit a "piano-roll". This time however, we use the image in Figure 6 as our input. The interpretation of the image is exactly the same as in the score for Mary Had a Little Lamb; when a dark pixel is encountered, a sine wave is played. We can still listen to this signal, though doing so is unpleasant. For the sake of space, we will show only four stages of the radio pipeline: the original source, the modulated signal, the received signal immediately before it is demodulated, and the final, filtered signal. These stages correspond to Figure 4 (a and b), and Figure 5 (d and f).

Figure 6. Original image.

Figure 7 shows these four image signals using a modulation index which we found empirically to produce the best quality transmission. One can see the ghosts of the original image in the modulated signal; these are often referred to as significant sidebands, and they correspond to the overtones we saw in our earlier visualization. The demodulated signal is a clear and accurate recreation of our original image.

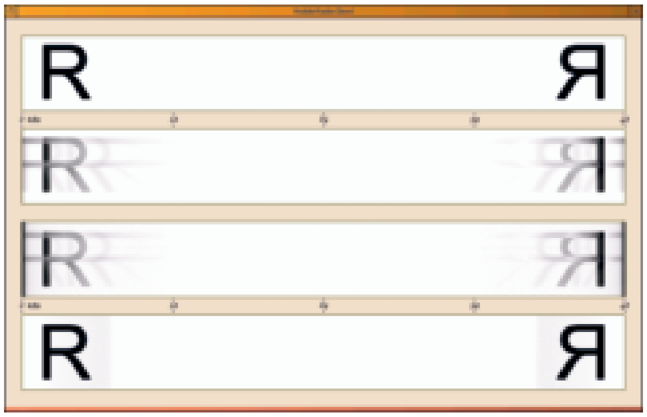

Figure 7. Illustration of an adequate modulation.

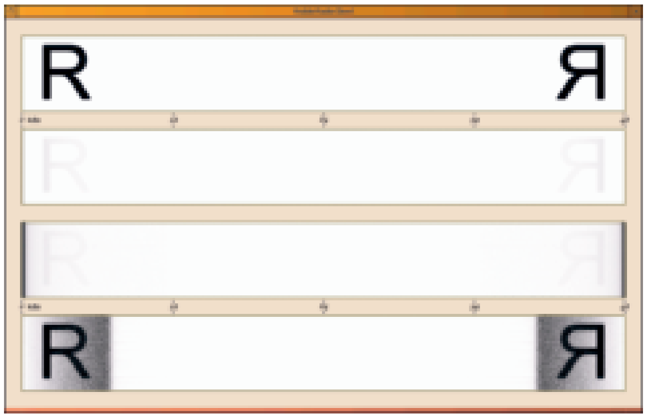

Figure 8. Illustration of insufficient modulation.

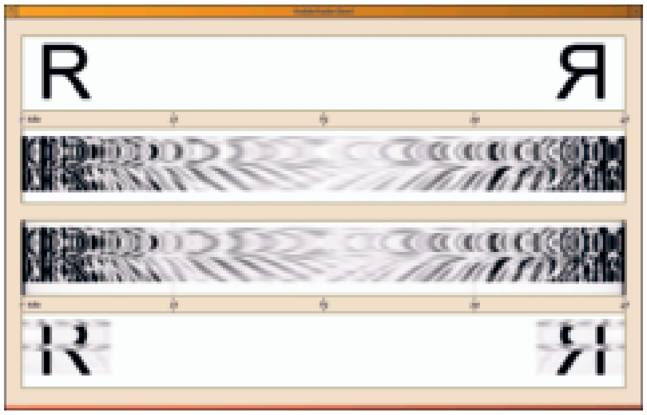

Figure 9. Illustration of overmodulation.

In Figure 8, we see what happens when the modulation index is reduced by a factor of 10. The modulated signal looks much like the original image except for the fact that it is dim and fades a bit on the right side. No other significant sidebands are present. If we look at the signal on the receive side, right before it is demodulated (Figure 8), we find that the intensity of the modulated signal is not significantly stronger than the intensity of the noise, and thus we should not be surprised that the demodulated signal is quite noisy.

Increasing the modulation index does increase the noise immunity, but the modulation index cannot be increased without bounds. Figure 9 shows what happens when we increase the modulation index by a factor of 10. For purposes of illustration, we have removed the first low pass filter from the receiver and transmitted the signal under low noise conditions. With the filter in place, the central portion of the modulated signal would be discarded, resulting in even more distortion of the demodulated result. Even without the filter, the high modulation index leads to a large number of significant sidebands. The higher frequency sidebands begin to interfere with their symmetric counterparts which we remarked upon in Section 3.3 above. As is evident in Figure 9, when this occurs, the signal can no longer be demodulated successfully.

4 DISCUSSION AND FUTURE WORK

Studying RF engineering concepts has always been a difficult task for students as it involves advanced mathematics and a lot of hands-on experiments. Students have to validate the mathematical equations or derive new relations based on experimental observations. The use of various software-based simulation tools, such as Matlab provides a great deal of help, but these tools are not as intuitive as one might wish. On the other hand, the visualization tool presented in this paper allows linking an implementation of a real RF design with a familiar visual metaphor that is informative and easy to understand. We are investigating how to include other forms of visualization, and especially interaction, to make the current tool more complete and engaging to use.

The proposed process visualization approach is based on the idea of visualizing spectral data in order to understand the process. We transmit images encoded as spectral data so that visual patterns are easily perceptible. In our current implementation, we focus on the qualitative data, and reduce the quantitative data. However, the latter is important for judging the quality of performance of individual components in the SDR pipeline. We are investigating how to integrate this type of data into the plots in a way that can enhance the visualization. For instance, we have investigated the use of color in our visualizations for such a purpose. Figure 10 shows an early example in which what is perceived to be signal noise is shown in gray whereas the actual signal is shown in color. We carefully adjust the border between the brightest gray level corresponding to the highest noise value and the darkest shade of blue corresponding to the lowest level of a useful signal so that the noise does not appear more intense then the signal. Such a color-enhanced spectrogram provides a more intuitive illustration of what constitutes the signal and what belongs to the noise floor and what portion of the spectrum is occupied by the useful signal vs. noise whereas this distinction is not clear in Figure 5b. We are investigating what other signal characteristics can be mapped into color as well as the use of other color scales.

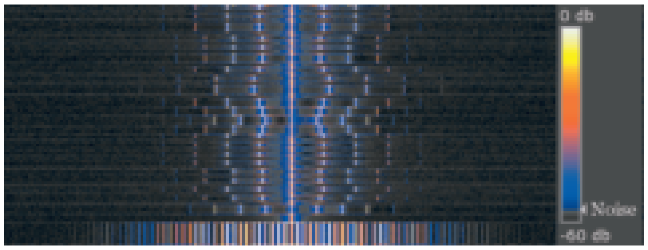

Figure 10. Experimenting with color.

We are interested in creating a more metaphorical 3D visualization that will convey the narrative of the radio's operation, particularly to an uninitiated audience, and will enable to embed more information about the radio operation than the 2D functional block diagram. We have developed a 3D Visible Radio client, called SDRVis, which is a refinement of the functional block diagram. In SDRVis, the edges representing links between blocks are widened into broad paths, upon which the spectrograms scroll from source to destination (Figure 11). Wedges represent DSP primitives, and correspond to the blocks of the functional block diagram. Signals in intermediate blocks are buffered, so that a signal appears to scroll into a functional block, be modified, and its output scrolls out, rather than appears at the output of all stages simultaneously. One may navigate closer to areas that one wished to inspect; Figure 12 shows a close-up of the demodulation block of the receiver. Note how shadows in the background allow the viewer to see the functional block diagram of the SDR at all times. Figure 12 also shows that the Visible Radio is also able to transmit and receive grayscale images with a good degree of fidelity.

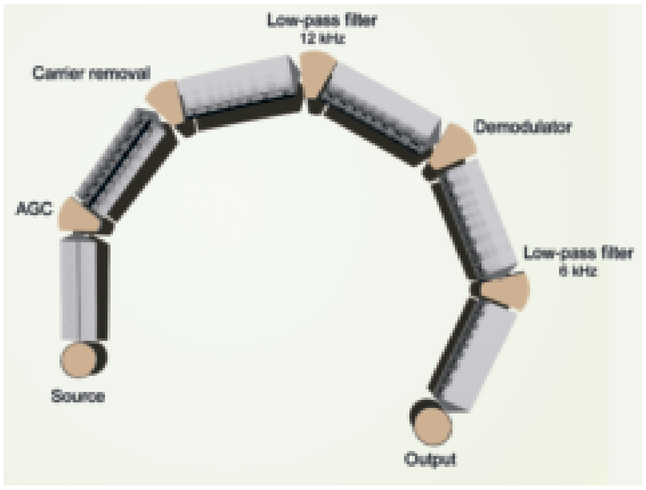

Figure 11. NCSA GNU Radio FM receiver in 3D.

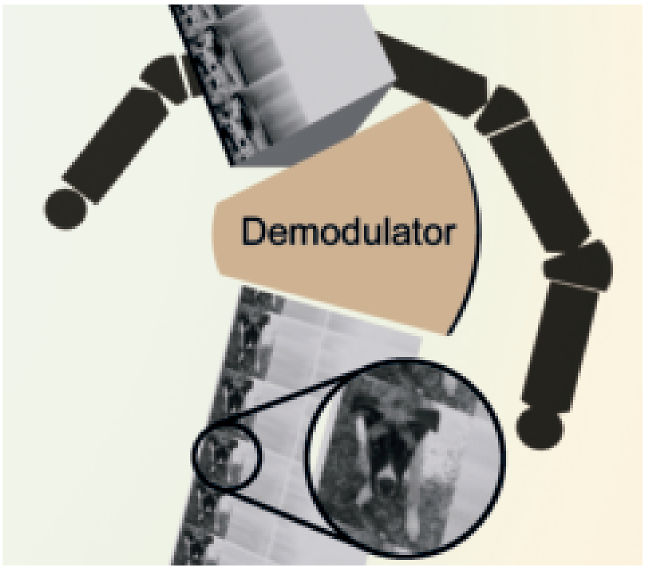

Figure 12. A detailed view of the demodulator node.

The 3D Visible Radio was created using a 3D animation program called Maya. A custom Maya plug-in performs the layout of the 3D block diagram automatically at start-up and allows buffered signal data to be sent to the real-time visualization client over a socket.

Besides serving as a good educational tool, the existing visualization software has other applications. However, current implementation requires a manual inclusion of data taps in the SDR code and a manual setup of the visualization client. We are investigating how the SDR code can be automatically instrumented with the data probing taps and how the visualization client can reconfigure itself for a new chain of SDR processing blocks. This is particularly of interest with the latest version of the GNU Radio code that allows a dynamic modification of the data processing pipeline.

So far we have implemented FM radio visualization with only one parameter adjustable (FM modulation constant). We are investigating how other radio characteristics can be controlled interactively and their effects on the radio performance visualized. We are also looking into implementing other types of radio designs, such as Amplitude Modulated (AM) radio, data modem, etc. An assortment of such radio modalities available for visualization could be a useful tool in studying and comparing RF designs by the EE students. We are interested in integrating our Visible Radio project into introductory RF and DSP design courses offered to the EE students. This, however, will require a development of a more comprehensive SDR visualization toolkit.

5 Acknowledgements

We thank both Dr. Paige Scalf and Stuart Levy for comments on earlier versions of this manuscript. This work was performed at the National Center for Advanced Secure System Research (NCASSR) and funded by the Office of Naval Research (ONR) grant N00014-3-1-0765.

References

[1] J. Reed. *Software Radio: a Modern Approach to Radio Engineering*. Prentice Hall PTR, Upper Saddle River, NJ, 2002.

[2] W. Tuttlebee (ed.). *Software Defined Radio: Origins, Drivers and International Perspectives*. John Wiley & Sons, Ltd., West Sussex, UK, 2002.

[3] E. Blossom. GNU radio: tools for exploring the radio frequency spectrum, *Linux Journal*, 122, 4, June 2004.

[4] E. Blossom. Listening to FM radio in software, step by step. *Linux Journal*, 125, 1, Sept. 2004.

[5] A. Betts, M. Hall, V. Kindratenko, M. Pant, D. Pointer, V. Welch, P. Zawada. The GNU software radio transceiver platform. In *Proceedings of SDR Technical Conference*, volume C, pages 41-46, November 2004.

[6] G. Youngblood. A Software Defined Radio for the Masses, Part 1. *QEX*, 213, 13-21, 2002.

[7] G. Youngblood. A Software Defined Radio for the Masses, Part 4. *QEX*, 217, 20-31, 2003.

[8] M. Frerking. *Digital Signal Processing in Communication Systems*. Van Nostrand Reinhold, New York, NY, 1993.

[9] R. Lyons. *Understanding Digital Signal Processing*. Prentice Hall PTR, Upper Saddle River, NJ, 1997.