



34th

ARRL and TAPR DIGITAL COMMUNICATIONS CONFERENCE

Chicago, Illinois

October 9-11, 2015





34th ARRL and TAPR DIGITAL COMMUNICATIONS CONFERENCE



ARRL

225 Main Street
Newington, CT 06111-1494 USA
tel: 860-594-0200 www.arrl.org



Tucson Amateur Packet Radio

PO Box 852754
Richardson, TX 75085-2754 USA
tel: 972-671-8277 www.tapr.org

Copyright © 2015

The American Radio Relay League, Inc.

Copyright secured under the Pan-American Convention

International Copyright secured

All rights reserved.

No part of this work may be reproduced in any form except by written permission of the publisher. All rights of translation reserved.

Printed in USA.

Quedan reservados todos los derechos.

ISBN: 978-1-62595-040-6

First Edition

Copies of this book can be ordered from **www.lulu.com**.

Welcome!

The ARRL/TAPR Digital Communications Conference is the premier gathering of Amateur Radio digital enthusiasts in the country, if not the world. This year we welcome everyone to Chicago for the 34th meeting since these annual conferences began.

As anyone who has ever attended a Digital Communications Conference will tell you, these gatherings are excellent venues for innovative ideas and discussions. Within these proceedings, for example, you'll find papers on topics ranging from HF receiver testing to ARDOP, the Amateur Radio Digital Open Protocol. Most of these papers are appearing in public for the very first time.

The ARRL thanks Tucson Amateur Packet Radio (TAPR) for all the hard work they do to make these conferences successful. Were it not for TAPR, it is possible that the conferences would not occur at all.

Dave Sumner, K1ZZ

ARRL Chief Executive Officer

September 2015

Table of Contents

| | |
|--|-----|
| QRPi – A Raspberry Pi QRP TX Shield Design; Zoltán Dóczy, HA7DCD | 1 |
| VOA Radiogram: Text and Images via Shortwave Broadcasting; Kim Andrew Elliott, KD9XB | 10 |
| HF Receiver Testing: Issues & Advances; Adam Farson, VA7OJ/AB4OJ | 20 |
| The AREDN Project (AREDN.org); Andre Hansen, K6AH | 35 |
| Feher Modulation 16 QAM; Patrick Jungwirth, PhD | 49 |
| Update on DATV-Express Exciter for Digital-ATV; Ken Konechy, W6HHC | 59 |
| Measuring the Ionosphere at Vertical Incidence using Hermes, Alex, and Munin Open HPSDR and Gnuradio; Tom McDermott, N5EG | 74 |
| Arduino CAT Controller for HPSDR; John Melton, GØORX/N6LYT | 87 |
| ARDOP (Amateur Radio Digital Open Protocol): A next generation digital Protocol for HF and VHF/UHF; Rick Muething, KN6KB, Matthew Pitts, N8OHU, and John Wiseman, GM8BPQ | 98 |
| An OS Independent and Device-Independent Mobile Web Front Panel for Radio Transceivers; Bruce Perens, K6BP | 105 |
| Broadband-Hamnet™; Patrick Prescott, KC1AJT | 114 |
| OpenWebRX: SDR Web Application for the Masses; Andras Retzler, HA7ILM | 122 |
| Modulation – Demodulation Software Radio; Alex Schwarz, VE7DXW, and Guy Roels, ON6MU | 130 |
| Design of a Practical Handheld Software Radio: Part II; Chris Testa, KD2BMH | 144 |
| A Radio Server for VHF+ Contesting and Weak Signal Work; Phil Theis, K3TUF | 155 |
| SatNOGS: Satellite Networked Open Ground Station; Daniel J. White, PhD, ADØCQ | 172 |

QRPi – A Raspberry Pi QRP TX Shield Design

Zoltán Dóczi, HA7DCD
Budapest, Hungary
+36 70 316 81 56
zoltan@rfsparkling.com

Abstract

"Be Smart, Not Strong" this should be the self explaining phrase of the QRP term in amateur radio. Low power operation is always more difficult than using hundreds or thousands of watts RF power. But the smile on your face after the first thousands miles long QSO, using portions of one watt is worth the challenge! QRP enthusiasts instead of spending time and money on increasing power capabilities of its station prefer a smarter way: to learn about new modulations and coding techniques and applying them in everyday HAM operation practice.

Nowadays one of the most impressive QRP mode is Joe Taylor, K1JT's [8] WSPR [9] (pronounced "whisper"). WSPR stands for "Weak Signal Propagation Reporter". Programs written for WSPR mode designed for sending and receiving low-power transmissions to test propagation paths on the MF and HF and recently UHF bands. Users with internet access can watch results in real time at wspn.net.org

The QRPi board (or shield as referred by the community today) is an inexpensive way turning a Raspberry Pi single-board computer into a QRP transmitter.

Keywords: QRP, RPi, SDR, WSPR, open-source

Introduction

My QRPi shield is inspired by the WsprryPi [10] open source program, I've started to play with it as any other HAM operator, experimenting with the WSPR mode. At the beginning I followed the available articles [2] and DIY [10] guides about connecting a Low Pass Filter (LPF) to the RF output pin (GPIO 4) of the RPi computer. As an enthusiastic RF engineer and HAM operator I was instantly measuring the output signal with a signal analyzer and found a broadband noise from 0 Hz up to several harmonics [Fig 1] That was obvious that a LPF solves only the harmonic content attenuation and doesn't help against the broadband noise of the RF signal synthesized with the BCM2835's "General Purpose GPIO Clocks".

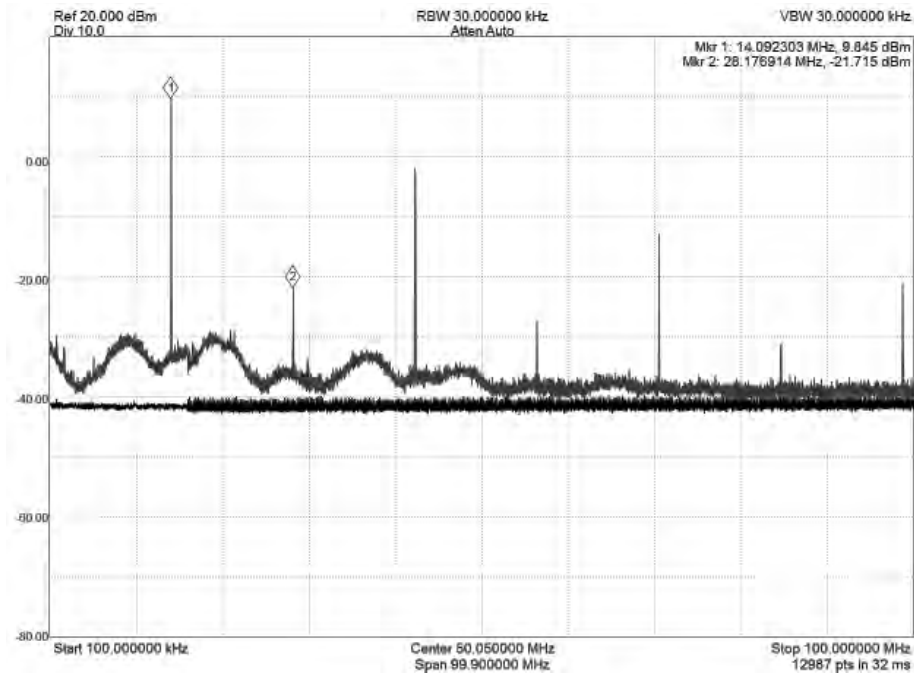


Figure 1. - RF Output spectrum of RPi's GPIO4 pin without filtering

At that point I made a research to find the possible inexpensive but efficient way to filter out the noise around the carrier. Lew Gordon's excellent article [3] led me to start my circuit simulations and to build my early prototypes based on his Band Pass Filter (BPF) advice. After successfully optimizing the BPF [Fig 3] values considering the parasitic parameters of the applied SMD inductors I saw a great improvement at the output spectrum [Fig 2].

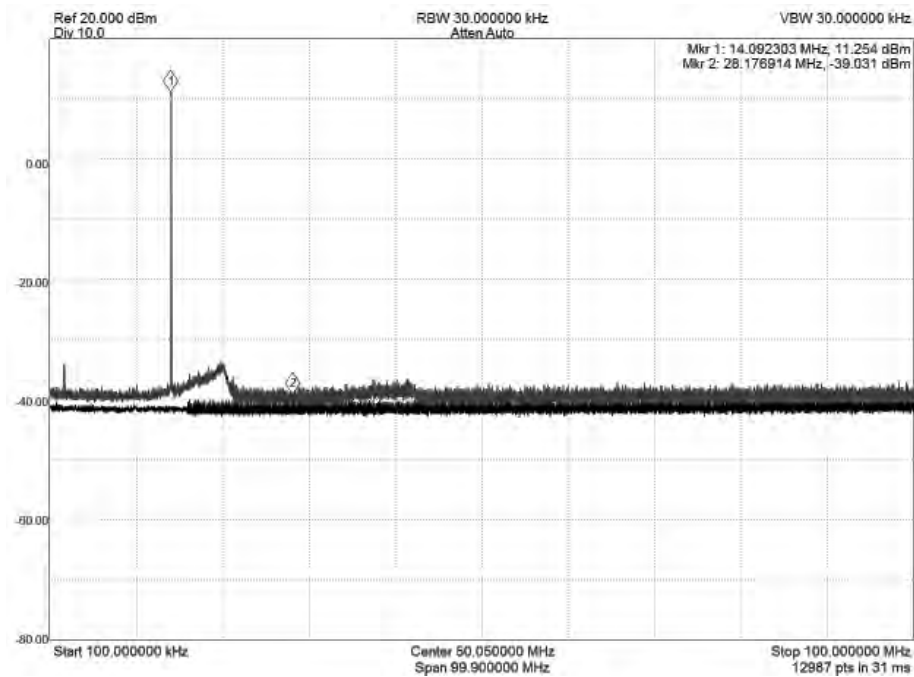


Figure 2. - RF Output spectrum of RPi's GPIO4 pin after BPF

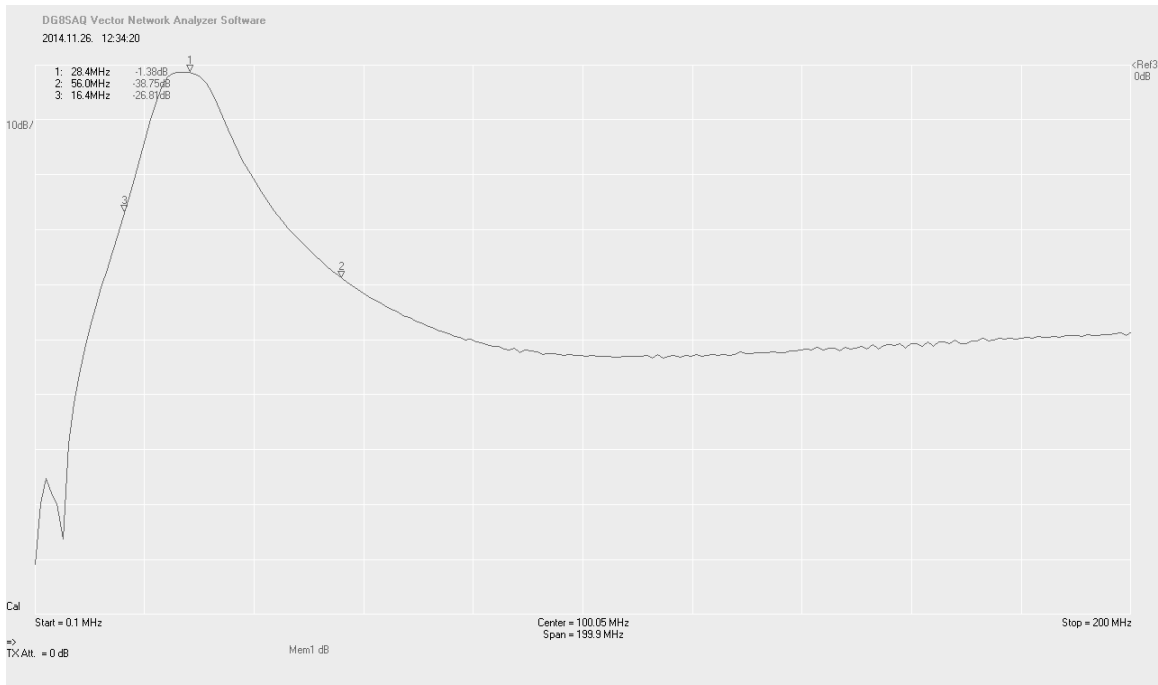


Figure 3. - Frequency response of the 10m, 3 element BPF on a VNA

At that stage of the design the harmonics filtered by the LPF, and the broadband noise filtered by the BPF were both acceptable. However there were still one thing missing: no buffer stage to protect the BCM2835 SoC's clock generator output stage. Hardware failure due to the unbuffered operation of the WsprryPi program was reported by a few HAM operators, possibly due to overload from nearly broadcast transmitter stations. If buffer amplifier was already needed it was a good idea to add some gain to the system. Eventually using a single FET amplifier stage [fig 7] 10 dB gain achieved, delivering +20 dBm output power at the end of the LPF [fig 4].

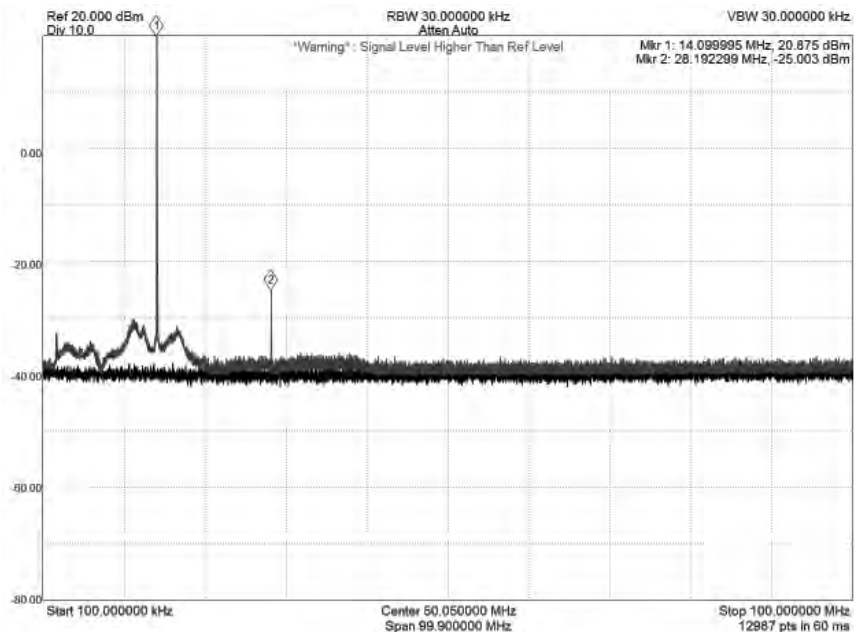


Figure 4. - RF output spectrum of RPI's GPIO4 pin after BPF + PA + LPF

For ESD and static discharge protection an ESD suppressor diode was added to the antenna terminal of the circuit.

I've targeted the absolutely smallest and most compact form factor. I've seen several RPi HAM accessories which were too "bulky" in my point of view. Using large external PCBs with long cables attached to the RPi it's destroy the true value of the card sized computer: small, mobile and flexible.

As I wanted to give an inexpensive QRP shield for the HAM and RPi community, mass production capability (SMD parts) and cheap component selection (eg. no SMA connector) was mandatory from the initial planning phase.

Regarding compactness I've exploited the advantages of inside PCB milling, leaving a gap for the RPi's LCD ribbon cable connector. This way the QRPi shield has low profile while sitting on the RPi, not even the highest point of the card-sized computer. This hopefully enables the use of popular stock RPi plastic enclosures with the QRPi.

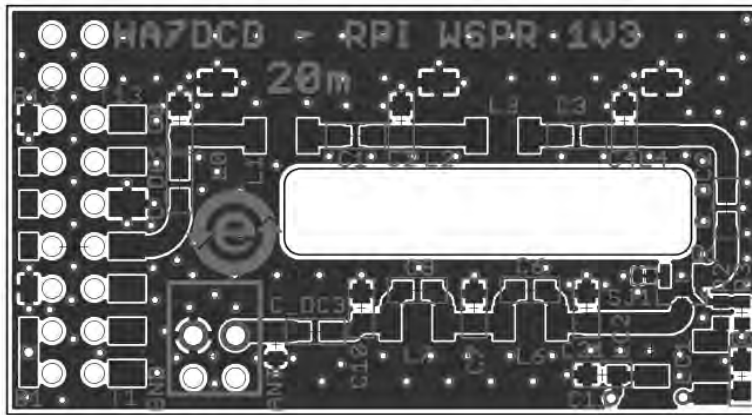


Figure 5. - CAD Layout screenshot of the QRPi shield

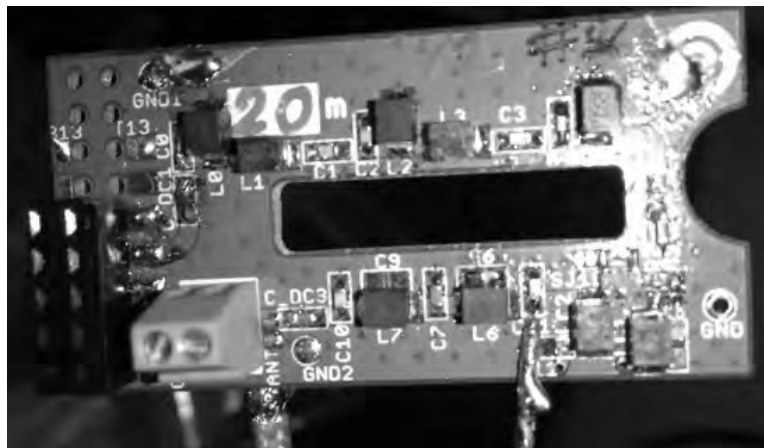


Figure 6. - Close up of a QRPi prototype PCB

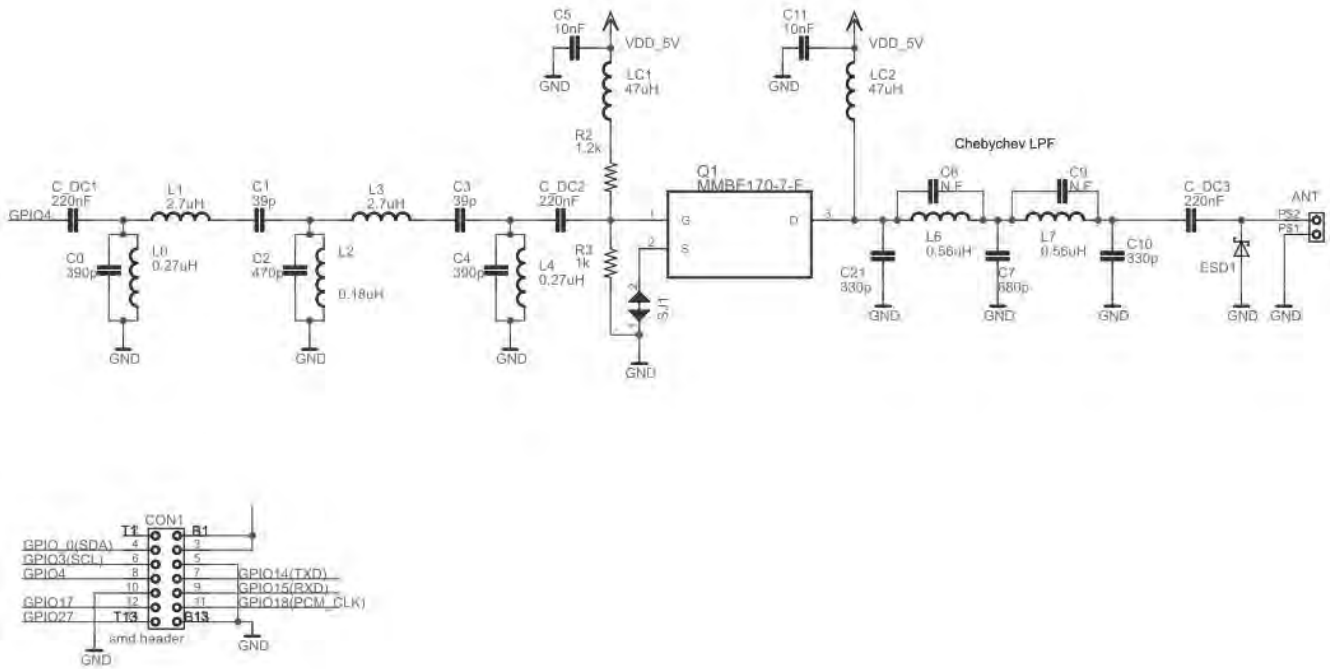


Figure 7. - Schematic of the QRPI shield

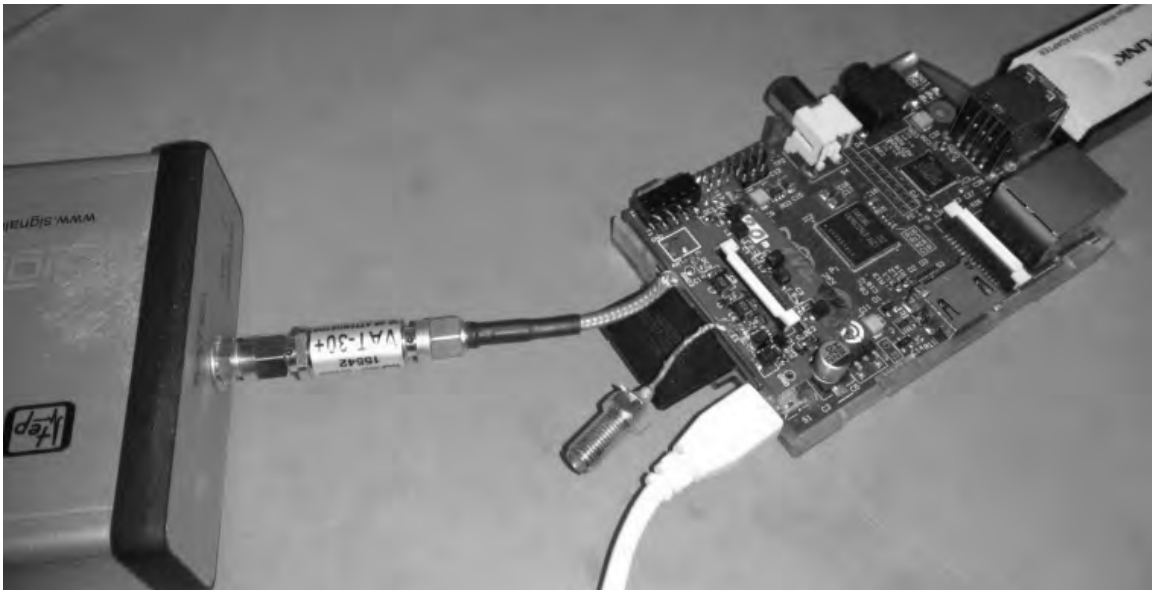


Figure 8. - Device Under Test on the spectrum analyzer

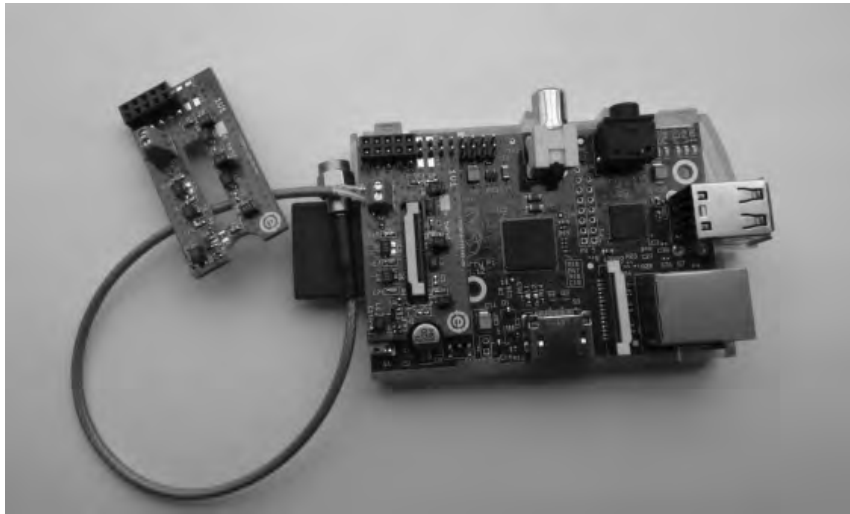


Figure 9. - QRPi on a Raspberry Pi computer

QRPi WSPR field tests

Laboratory measurements and fine tuning is one thing, another important factor is the real life operation and feedbacks from beta testers. Measurement and reports accumulated using WsprryPi and the QRPi shield since December 2014 till nowadays on daily basis using the latest prototype [fig 9]. Without any sophisticated antenna, using only a simple outdoor random wire at 2m height 1000...2000 km QSOs are typical on the 10 and 20m band with the +20dBm output power [fig 10-11].

Until the release of this paper the following digital modes and tools were tried and measured using QRPi:

- WSPR TX [10] - laboratory and field tests
- WSJT [11] - due to lack of resources not tested yet by the author of this paper, but seen feedbacks from HAM operators who use this with RPi
- CW TX [12] - laboratory tests
- SSTV TX [13] - laboratory tests
- Signal Generator tool [14] - laboratory tests



Figure 10. - HA7DCD - LA9JO, 2400km 14.097185 MHz, WSPR



Figure 11. - Several European stations copying QRPI WSPR beacon, WSPR

Possible further developments

However the BPF filter does the job when filtering broadband noise coming from the BCM2835 SoC, there are still issues when amplifying the relatively wide band RF signal. From my measurements and investigations I realized that the FET amplifier starts to work as a mixer when the CW signal and the broadband noise filtered portion driven to its gate. The mixing product can be observed at the spectrum analyzer screenshot [fig 12], ranging from 0 Hz to 10 MHz on the left side. I've successfully simulated the QRPi mixing product behavior using noise signal from a signal generator, or driving two sinus signals on the gate of the FET.

Investigation with FET biasing trials doesn't show a solution for this. The spurious content is still at an acceptable level [fig 4] however for upcoming revisions considering another PA structure with less mixing behavior might be a good idea.

From the software side a possible workaround should be to understand BCM2835's clock generators more deeply. Fine tune it and decrease noise.

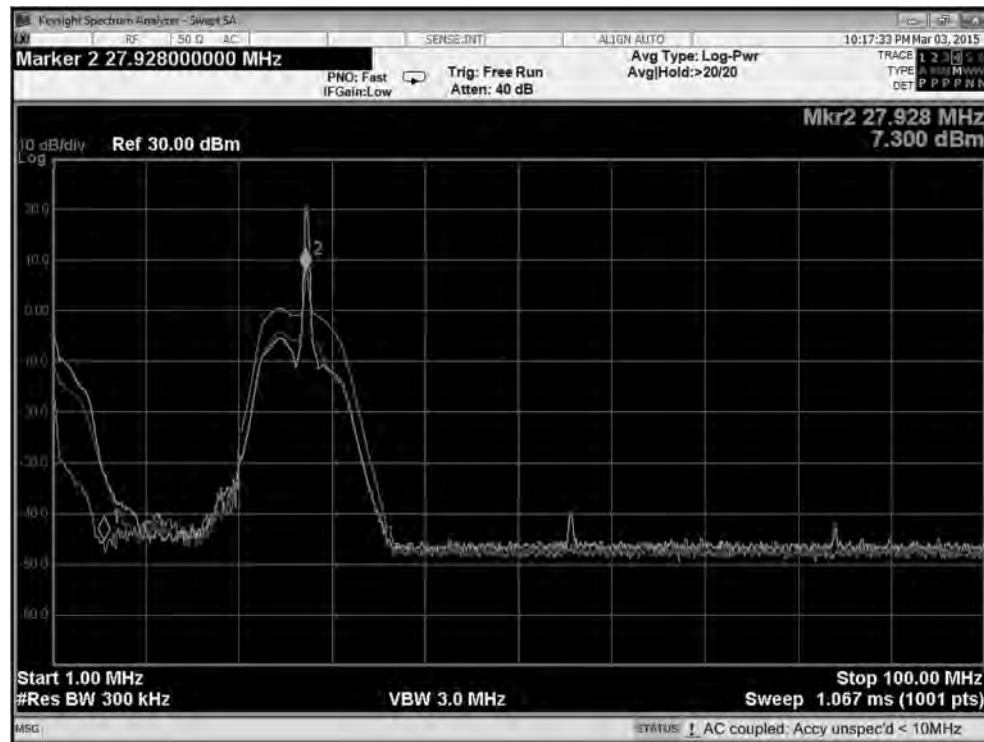


Figure 12. - FET amplifier acting as a mixer because of wide band noise

I would like to say thanks for the kind help of:

Steven Bible - N7HPR

Andris Retzler - HA7ILM

Hackerspace Budapest and András Veres-Szentkirályi "dnet" - HA5KBP

Tucson Amateur Packet Radio Corp

HA5KFU Schönherz Amateur Radio Club

Reference List

1. QRP definition - https://en.wikipedia.org/wiki/QRP_operation
2. TUTORIAL: TRANSMITTER (PA) OUTPUT FILTERS by Paul Harden, NA5N
http://www.aoc.nrao.edu/~pharden/hobby/lpf_pa.pdf
3. Band Pass Filters for HF Transceivers by Lew Gordon, K4VX
<http://www.arrl.org/files/file/Technology/tis/info/pdf/8809017.pdf>
4. Raspberry Pi website - <https://www.raspberrypi.org>
5. Raspberry Pi Low Level Peripherals - http://elinux.org/RPi_Low-level_peripherals
6. Raspberry Pi SoC, Broadcom BCM2835 - <http://www.raspberrypi-projects.com/pi/pi-hardware/bcm2835>
7. BCM2835 datasheet - <http://www.farnell.com/datasheets/1521578.pdf>
8. K1JT website - <http://physics.princeton.edu/pulsar/K1JT/>
9. Weak Signal Propagation Reporter Network - <http://wsprnet.org>

Raspberry Pi transmitter programs working with QRPI TX shield:

10. WSPR - <https://github.com/JamesP6000/WsprryPi>
11. WSJT - <http://hajos-kontrapunkte.blogspot.hu/2014/04/silent-whisper-jt9-on-cubie-truck.html>
12. CW - <https://github.com/JamesP6000/PiCW>
13. SSTV - <https://github.com/JennyList/LanguageSpy/tree/master/RaspberryPi/rf/sstv>
14. Signal Generator -
https://github.com/JennyList/LanguageSpy/tree/master/RaspberryPi/rf/freq_pi
15. HA5KFU Radio Club - <http://ha5kfu.sch.bme.hu>
16. Hackerspace Budapest - <http://hsbp.org/>

VOA Radiogram: Text and Images via Shortwave Broadcasting

Kim Andrew Elliott, KD9XB
Voice of America
330 Independence Avenue SW
Washington, DC 20237
ke@bbg.gov

Abstract: The Internet has largely replaced shortwave radio for the broadcast of news and information across international boundaries. A growing number of countries, however, are blocking Internet content from abroad. As a possible workaround, digital text modes familiar to the amateur radio community can be used to broadcast news via existing shortwave transmitters and can be received on any shortwave radio, but software is required to decode the text. VOA Radiogram is a weekly Voice of America program experimenting with text and images through a shortwave broadcast transmitter

Keywords: broadcasting, HF, shortwave, MFSK

Introduction

International broadcasting services such as the Voice of America and BBC World Service traditionally employed shortwave radio to transmit news and information across national boundaries. In recent years, international broadcast content has shifted to the Internet as more audiences have access to this new medium. The Internet enables not only audio, but also text, images, and video to be conveyed over long distances, and it provides the audience with more control over the choice of content, as well as an immediate means of feedback.

Unlike shortwave radio, Internet content is usually brought into a country via landlines, and is routed through Internet providers in the “target” country. These factors provide a national government the opportunity to block Internet content it deems undesirable. Circumvention technologies, e.g. Psiphon and Tor, afford audiences in the target country some opportunity to overcome this interdiction, but an even larger industry (much of it based in the United States) help governments step up their counter-circumvention efforts.

Internet content can be disrupted not only by dictators, but also by disasters, both natural and caused by humans.

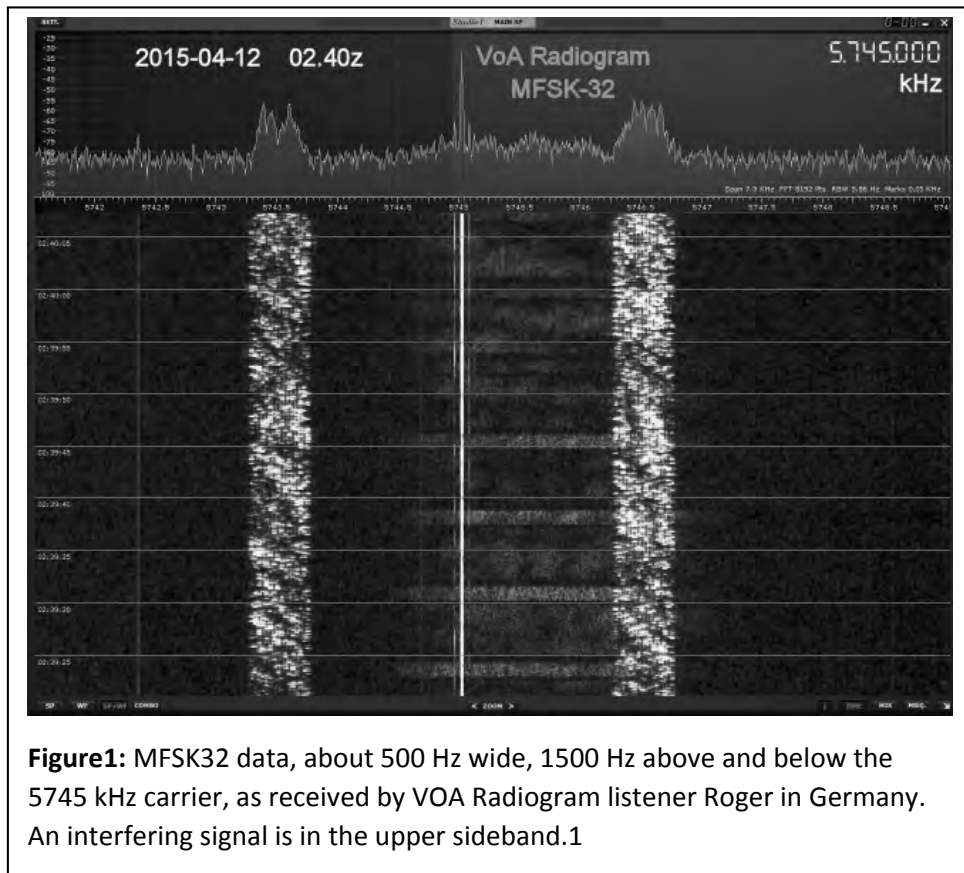
Digital modes via analog broadcast

In the past decades, while pondering solutions to Internet interdiction, I also became active in the amateur radio digital modes. I was most impressed by the ability of the digital modes to transmit text successfully even in the worst reception conditions. This led me to wonder if the digital text modes from amateur radio could be used on a shortwave broadcast transmitter.

Educated by my engineering colleagues at the International Broadcasting Bureau (parent agency of the Voice of America), I learned that this could be done, at least in theory. In fact, the modes could even

be transmitted in the AM mode of the IBB's shortwave broadcast transmitters, and received on typical low-cost shortwave radios lacking single sideband capability. Early tests used amateur radio transceivers with a dummy load "antenna" to nearby shortwave radios. By the spring of 2012, brief transmissions via private shortwave broadcast stations in the United States resulted in successful decodes hundreds of kilometers from the transmitter.

In March 2013, VOA Radiogram went on the air. This half-hour program is transmitted four times per weekend through a 50-year-old GE transmitter, operating at 80 kilowatts, at the Edward R. Murrow transmitting station near Greenville, North Carolina. Two of the transmissions are via a curtain antenna directed to Europe, and two are via a dipole to the Caribbean. The broadcasts are typically heard both in and outside their nominal target areas.



Most of the content on VOA radiogram consists of science news from the Voice of America website, voanews.com. Text from the website is pasted to the transmit pane of Fldigi, the well-known digital encoding/decoding software from authored by David Freese, W1HKJ. Fldigi transforms the text to the tones which are inserted into a digital audio file for broadcast. Most listeners use Fldigi to decode, but some decode with other software, such as MultiPSK and DM780.

Thousands of reports have been received from shortwave listeners and radio amateurs throughout Europe and North America, as well as some in Latin America, and, beyond the nominal coverage area of the North Carolina transmitter, in Asia and the Pacific. Listeners use a variety of equipment to

receive VOA Radiogram, including amateur transceivers, shortwave portables, antique radios with shortwave bands, and SDR black boxes and dongles. Reception on inexpensive radios is



Figure 2: Typical equipment needed to receive and decode VOA adiogram broadcasts is a portable shortwave radio and a notebook PC, with a patch cord feeding audio from the former to the latter. Appropriate software is also necessary.

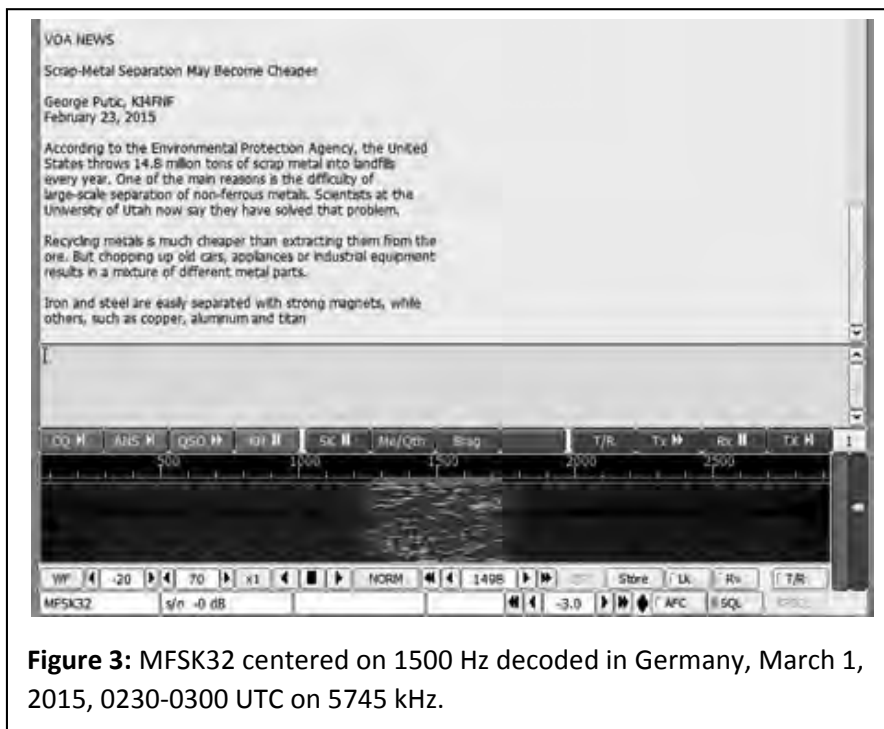


Figure 3: MFSK32 centered on 1500 Hz decoded in Germany, March 1, 2015, 0230-0300 UTC on 5745 kHz.

especially encouraged. Audio is typically fed to a computer using a patch cord, or through an interface such as Signalink. “Acoustic coupling,” in which the built-in microphone of a laptop computer is placed near the radio’s speaker, is not uncommon.

MFSK and challengers

During the early weeks of VOA Radiogram, modes of similar speeds were transmitted to compare the number of errors. The MFSK modes performed well early on, so, in the manner of boxing, “champion” MFSK took on challengers one at a time. One week the 110 word per minute PSKR125 would be followed by the 120-wpm MFSK32, and the 220-wpm PSKR250 would be followed by the 240-word MFSK64. In the subsequent weeks, MFSK modes would be compared to similar mode speeds of MT-63, DominoEX, and THOR. Throughout this process, the MFSK exhibited fewer, and at least no more, errors than the competing mode.

With MFSK established as the most promising mode, VOA News content was transmitted in different speeds of MFSK to determine which provide ideal performance in shortwave broadcast conditions. Based on hundreds of reports, it was determined that MFSK32, at 120 wpm, provides the best combination of performance and speed in the conditions experienced by most VOA Radiogram listeners.

The MFSK16 mode is slower but can be useful in difficult reception conditions. For reliable transmissions paths, such as the typical distance between an IBB shortwave relay site and most VOA target countries, MFSK64 will usually succeed and allows twice the content during the time period than does MFSK32. General guidelines for the use of MFSK modes under different conditions are summarized in Table 1.

| Table 1: Guidelines for the use of MFSK modes in different shortwave reception conditions | | |
|--|-------------|---------------------|
| Mode | Speed (WPM) | Expected Conditions |
| MFSK16 | 60 | Poor |
| MFSK32 | 120 | Fair |
| MFSK64 | 240 | Good |
| MFSK128 | 480 | Excellent |

During the first year of VOA Radiogram, it was also discovered that text via analog transmitter provided more reliable reception than voice via analog transmitter, i.e. the original purpose of these transmitters. In reception conditions where voice content may be difficult to comprehend, text often provides a 100% decode. Text via shortwave extends the useful range of a shortwave broadcast transmitter.

Another advantage of text via analog radio is the opportunity for “unattended reception.” Text content can be received during the overnight hours and read after waking in the morning. Or the text can be receiving during the day, to be read when returning home from work.

If the Fldigi software is configured for the UTF-8 character set, and the user’s operating supports the language, alphabets with diacritics and even non-Latin characters can be accommodated. VOA Radiogram has successful transmitted content in Russian, Chinese, Tibetan, and even the right-to-left Persian language. This is obviously a vital feature in international broadcasting.

ལྷ་མོ་གཞུང་གིས་དགའ་པོ་ལྷན་སྐྱེས་ཆེ་རྒྱལ་གྱི་ཡོད་པར་མ་ལྷོས་པར། རེས་གཟའ་ཉི་མའི་ཉིན་དམངས་གཙོ་

Figure 4: Tibetan text decoded by Merkouris in Greece, August 31, 2015, 1930-2000 UTC on 15670 kHz, using Fldigi.

Images via shortwave broadcast

A bonus of the MFSK mode is its ability to transmit images. Images can add to the meaning of a VOA News story, or enhance its credibility.

SSTV modes have been tested on VOA Radiogram. MFSK has proven more satisfactory in part because, unlike with SSTV, the size and shape of the image can be adjusted. In general, images are limited to 200x300 or 300x200 pixels so that their transmission does not occupy too much time. Images of the same size in MFSK16, 32, 64, and 128 require the same amount of time to transmit, but resolution sharpens as the baud rate increases. The higher baud rates, however, also increase the chances for interference, usually exhibited as lines in the decoded image. As with text, MFSK32 has shown itself to be the best compromise, in this case between resolution and resistance from interference.

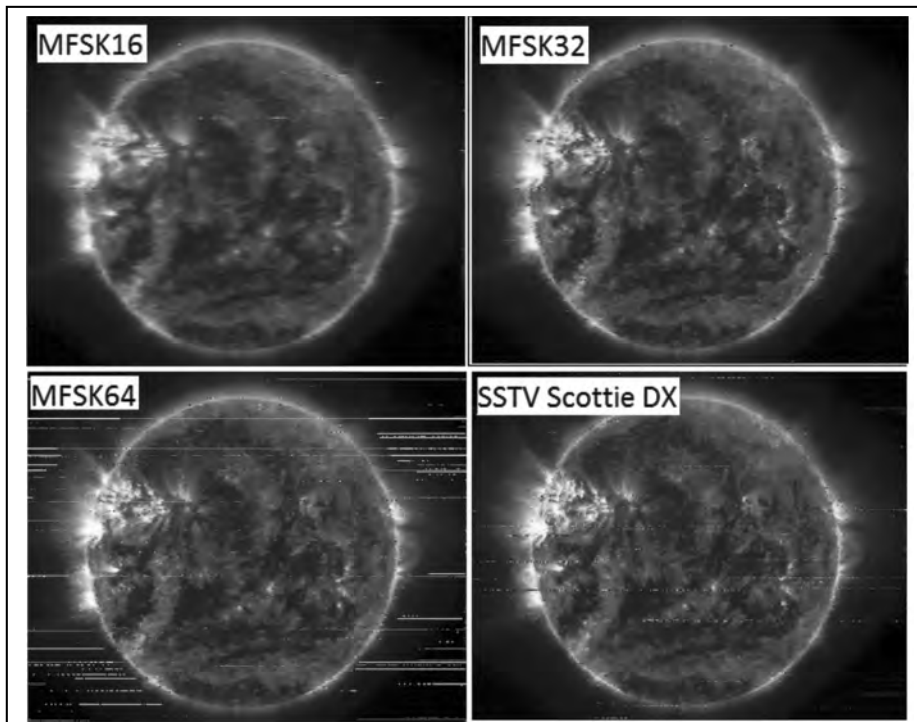


Figure 5: Four modes of images as received by Frank in the Netherlands, August 2, 2015, 1930-2000 UTC, on 15670 kHz.

Images are often fuzzy or noisy, the more important text content is usually received 100% thanks to the error correction built in to the MFSK modes.

Images in EasyPal, which uses DRM (Digital Radio Mondiale) encoding, have also been tested on VOA Radiogram. This is the all-radio version of EasyPal, not the hybrid version that refers the user to a server for download. As is typical of digital communication, the results were either perfect (and dazzling) or completely absent. As impressive as the successes were, the failures were too frequent to encourage continuation of the EasyPal experiments. The seven-minute transmission time for the most robust version of EasyPal images was another impediment.

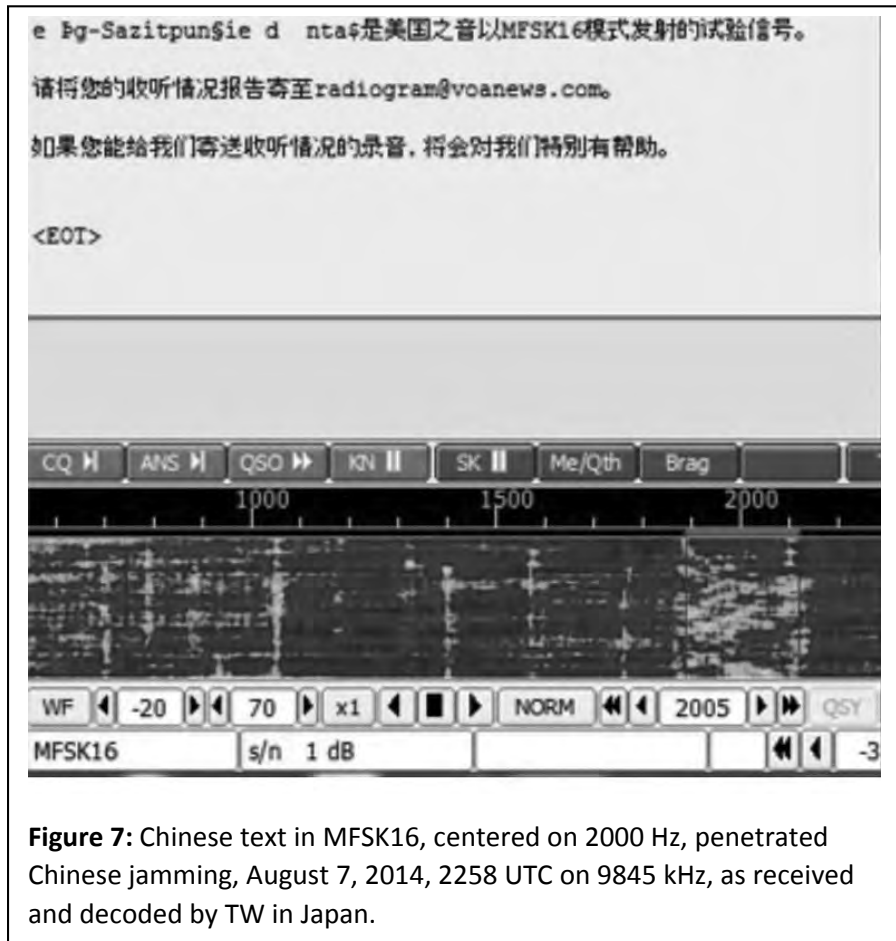


Figure 6: This EasyPal (DRM) image was decoded by Lorenzo in Italy, June 22, 2015, during the 1600-1630 UTC broadcast on 17860 kHz.

Jamming

If a regime blocks Internet content from other countries, there is a good chance it will also jam shortwave broadcasts from abroad. China vigorously jams shortwave broadcasts from the United States and other Western countries. Usually this is done by placing Chinese domestic radio programming, usually from more than one transmitting site, on the same frequency as a VOA or Radio Free Asia broadcast in Mandarin, Cantonese, Tibetan, or Uighur. Chinese operatic music or just noise is also used.

To determine if the text modes via analog shortwave broadcast can penetrate jamming, brief text transmissions were inserted in the shortwave broadcasts of the VOA Mandarin and RFA Cantonese services. The transmitters were the usual IBB relay facilities in Asia. MFSK16 and Olivia modes were used during these tests. Monitoring on receivers in Hong Kong and Japan demonstrated that these modes often, but not always, resulted in successful decodes of the Chinese text, despite conditions that prevented the comprehension of the accompanying VOA and RFA voice content.



Similar successful tests were conducted with Radio Martí shortwave broadcasts, which are heavily jammed by Cuba.

A future for text via shortwave broadcast?

The Fldigi software has remarkable capabilities, but it may be intimidating to non-technical users. Furthermore, audiences for international broadcasting do not need the encoding capabilities of this software. Wider accessibility to text and images via analog shortwave broadcast will require the development of software applications to simplify the decoding process and to enable decoding to be possible on mobile devices. (An Android version of Fldigi, AndFlmsg, is already available in beta version.)

It would also be very helpful for some shortwave receivers to include the ability to decode text modes. Such a development would be assisted by the adoption of one or a small number of modes to be used in shortwave broadcasting, and encouraged by the transmission of text by more shortwave broadcasters. (Text and images via analog radio can also be employed on the AM [medium wave] and FM radio bands.)

The development of easier-to-use software and hardware will probably not transform text via shortwave into a popular mass medium. If this technology can reach a few thousand users in a country

or area cut off from external Internet traffic, those users can then relay the information through what would become the country's intranet, or through non-electronic means in a disaster zone.

The future of shortwave broadcasting, voice or text, is certainly in question. As audiences have migrated to television and the Internet, many major shortwave broadcast transmitting facilities have been dismantled. Most developing countries have eliminated or largely curtailed their use of shortwave for domestic broadcasting now that their territories are covered by networks of FM and television transmitters. Fewer shortwave radios are available for sale; Sony, for example, is down to one model.

There is some interest in adopting DRM (Digital Radio Mondiale) to increase the fidelity of shortwave broadcasts. DRM is, however, less forgiving than analogue shortwave, with the audio dropping out completely in reception conditions, e.g. diminished signal strength and co-channel interference, that are not unusual on the shortwave bands. In similar difficult conditions, text via shortwave is *more* forgiving than analogue voice via shortwave. The availability of this technology to work around Internet interdiction may not be available if analogue shortwave transmitters and receivers are no longer available.

For more information about VOA Radiogram, including the transmission schedule, visit voaradiogram.net.

The author acknowledges the assistance of Gerhard Straub, K6XH, Director, Broadcast Technologies in the office of Technology, Services, and Innovation of the U.S. International Broadcasting Bureau, and Daniel Maxwell, N5LAY, IBB TSI electronic engineer, in the development of the VOA Radiogram project. And Macon Dail, WB4PMQ, as well as the staff at the IBB's Edward R. Murrow shortwave transmitting station near Greenville, North Carolina. Additional education and encouragement were provided by my amateur radio friends Christopher Rumbaugh, K6FIB, and Benn Kobb, AK4AV.



Figure 8: This \$25 radio in Germany provided audio for a successful decode of VOA Radiogram text.



Images received from VOA Radiogram, program 51, 22-23 March 2014



UTC: 0930 Sat 5745 kHz. 1600 Sat 17860 kHz. 0230 Sub 5745 kHz. 1930 Sun 5745 kHz. voaradiogram.net



New Zealand, Saturday 0930 UTC, 5745 kHz



Denmark, Saturday 1600 UTC, 17860 kHz



MFSK32 images received from VOA Radiogram, program 74, 30-31 August 2014, all via North Carolina

Virginia, Sunday 0230 UTC, 5745 kHz



Poland, Sunday 1930 UTC, 15670 kHz



voaradiogram.net

HF Receiver Testing:

Issues & Advances

(also presented at APDXC 2014, Osaka, Japan, November 2014)

Adam Farson VA7OJ/AB4OJ

Copyright © 2014 North Shore Amateur Radio Club

■9 October 2015

■HF Receiver Testing

■1

HF Receiver Performance Specs

- what HF operators "shop" for

- Sensitivity
 - Test signal level for a given signal/noise ratio in a given bandwidth
 - Usually stated as Minimum Discernible Signal (MDS) or noise floor:
 - Input in dBm at 500 Hz bandwidth to raise audio output level by 3 dB
- Selectivity & shape factor
 - IF or detection bandwidth at -6 and -60 dB points on passband curve
- Reciprocal mixing dynamic range (RMDR)
 - Test signal level at a given offset from RX freq. to raise audio output by 3 dB, minus MDS
 - A function of local-oscillator phase noise
- 2-signal, 3rd-order IMD dynamic range (DR3)
 - Input power of each of two equal test signals at a given spacing and 500 Hz bandwidth to raise demodulated IMD product by 3 dB, minus MDS
- Blocking gain compression
 - Level of strong signal at a given offset from weak signal to reduce level of demodulated weak signal by 1 dB. RX tuned to weak signal; 500 Hz bandwidth.

■9 October 2015

■HF Receiver Testing

■2

More HF Receiver Specs

- also important in choosing a rig

- 2-signal, 2nd-order IMD dynamic range (DR2)
 - Input power of each of two equal test signals falling outside band under test at 500 Hz bandwidth to raise demodulated IMD product in band under test by 3 dB, minus MDS. Receiver tuned to band under test (typically 14 MHz).
 - Determines receiver's susceptibility to QRM from HF broadcasters
- Frequency stability
 - Drift measured in Hz or parts per million (ppm) over time, and over a temperature range if a variable-temperature test chamber is available
 - Not usually an issue with modern synthesized radios
- Inband IMD
 - Relative amplitude of either of two narrow-spaced test signals (typically spaced 200 Hz) and their associated IMD products, measured at audio output
 - Severe inband IMD causes listener fatigue
- Image & IF Rejection
 - An old problem returns in receivers with inband 1st IF

■9 October 2015

■HF Receiver Testing

■3

Main HF Receiver Impairments

- Intermodulation Distortion (IMD)
 - Odd-order IMD
 - Even-order IMD
 - IMD from multiple carriers approaches noise
- Reciprocal Mixing Noise
 - RF signal or noise mixes with LO phase noise
- Image Response, IF Leakage
 - RF signal or noise response at image freq. & IF
- Sensitivity/MDS is not an issue in modern receivers.
 - Below 21 MHz, the receiver noise floor is \approx 10 dB below band noise.

■9 October 2015

■HF Receiver Testing

■4

IMD:

intermodulation distortion

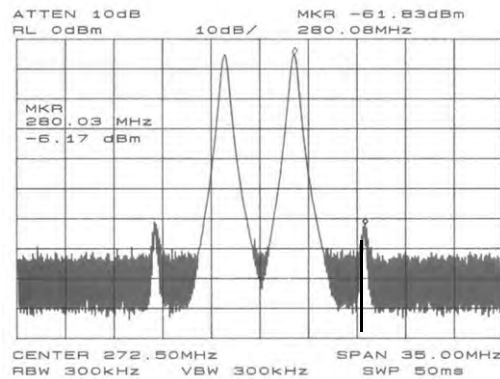
- Odd-order IMD
 - IMD products usually in same band as received signals f_1, f_2
 - 3rd-order IMD products: $2f_1 - f_2, 2f_2 - f_1$
 - Example: $f_1 = 7010$ kHz, $f_2 = 7015$ kHz. Products: 7005, 7020 kHz
- Even-order IMD
 - IMD products not in same band as f_1, f_2 .
 - 2nd-order IMD product: $f_1 + f_2$
 - Example: $f_1 = 8025$ kHz, $f_2 = 6010$ kHz. Product: 14035 kHz
- On a crowded band, multiple carriers generate a large number of IMD products
 - Limiting case is where spectrum of IMD products approaches Gaussian noise

■9 October 2015

■HF Receiver Testing

■5

IMD Example



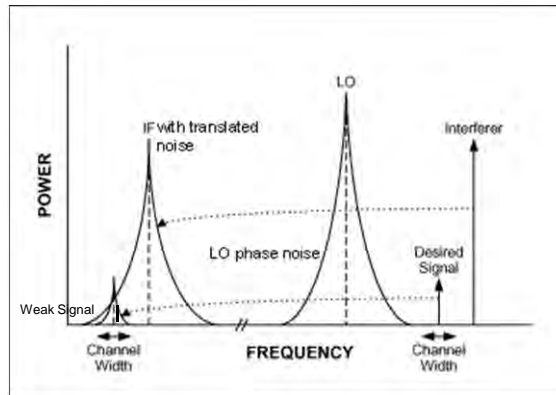
**IMD Example: $f_1 = 270$ MHz, $f_2 = 275$ MHz.
IMD products at 265 and 280 MHz.
280 MHz IMD product masks weak signal.**

■9 October 2015

■HF Receiver Testing

■6

Reciprocal Mixing Noise



Strong interferer mixes with LO phase noise to “throw” noise into IF channel. If the interferer consists of wideband noise, the IF channel will be filled up with noise.

■9 October 2015

■HF Receiver Testing

■7

Image Response, IF Leakage

- Image response:
 - Acceptance of signals at $f_0 \pm 2 * IF$ (f_0 = signal freq.)
 - Example: $f_0 = 10455$ kHz, $IF = 455$ kHz. Image: 10000 or 10910 kHz.
 - In modern receivers with high 1st IF, RF preselector suppresses image response almost completely.
- IF leakage:
 - Acceptance of signals at or close to 1st IF.
 - Example: 1st IF = 9 MHz. On 30m band, preselector may be sufficiently wide to pass some energy at 9 MHz. This will enter the IF chain and interfere with desired signals.
 - This is not a problem in receivers whose 1st IF is above the highest operating frequency.

■9 October 2015

■HF Receiver Testing

■8

Blocking and overload

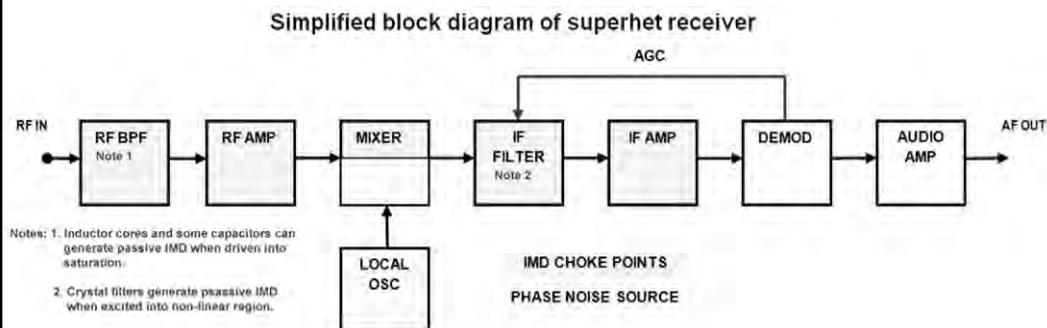
- Blocking: degradation of receiver sensitivity in the presence of a much stronger (blocking) signal.
- Blocking gain compression occurs when the interferer drives the first active RF stage to its compression point, thus causing desensing.
- Blocking gain compression is the difference in dB between the level of an incoming signal which will cause 1 dB of gain compression, and the level of the noise floor.
- Note that in a *direct-sampling SDR receiver*, no blocking occurs until the ADC is driven into saturation (clipping).

■9 October 2015

■HF Receiver Testing

■9

Typical Superhet Receiver showing impairment areas



- Multiple signals or wideband noise applied to RF IN will provoke IMD products at IMD choke points, and mix with LO phase noise to cause reciprocal mixing noise.
 - Steering diodes in RF/IF signal paths can also generate IMD.
- Passive IMD can occur in RF BPF components and crystal or mechanical filters.
- In addition to IMD and phase noise, image responses and IF leakage can arise if RF BPF is too wide to attenuate undesired signals at image frequency and IF.
- All these products will appear in IF/AF chain as added noise, spurs etc.

■9 October 2015

■HF Receiver Testing

■10

Issues in standard receiver test methods: *instrument limitations*

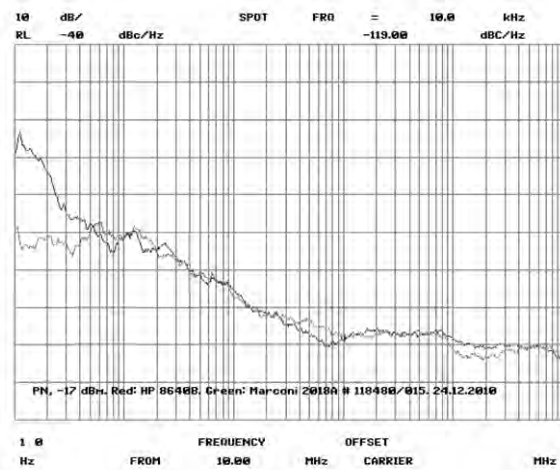
- Synthesized RF signal generators used for MDS, reciprocal mixing, blocking and IMD testing can have moderate to severe phase noise. This will degrade measurement accuracy.
 - A solution: ultra-low-noise crystal oscillators. These are costly and not frequency-agile. A vacuum-tube LC-type generator is also usable, but has poor frequency stability/accuracy.
 - Synthesized generators with excellent phase-noise performance are available, but are somewhat costly.
- Spectrum analyzers are frequently used for phase noise measurements.
 - Many high-end analyzer models support phase noise measurement software.
 - The limitation here is that the lowest phase noise value the instrument can display is that of its own internal phase noise.

■9 October 2015

■HF Receiver Testing

■11

Sig Gen Phase Noise Example *HP 8640B & Marconi 2018A*



■9 October 2015

■HF Receiver Testing

■12

Ultra-Low-Noise Crystal Oscillator



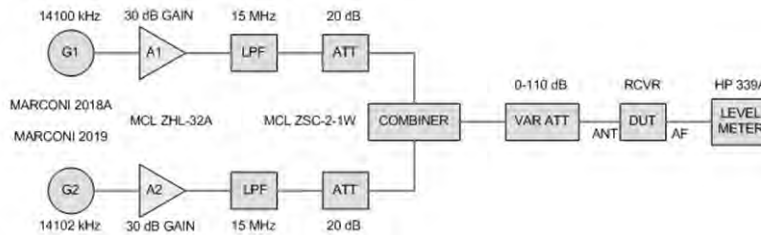
TYPICAL SPECS:
OUTPUT
 Frequency 5 MHz
 Level +13 dBm \pm 2dBm into 50 ohms
STABILITY
 Aging 1×10^{-6} per day
 after 30 days operating, typical
Phase Noise L (f)
 1 Hz -115 dBc/Hz
 10 Hz -145 dBc/Hz
 100 Hz -165 dBc/Hz
 1 kHz -176 dBc/Hz
Temperature Stability
 $\pm 5 \times 10^{-6}$, 0° to +60°C (Ref +25°C)

■9 October 2015

■HF Receiver Testing

■13

Typical 2-Tone IMD Test Setup: also used for blocking tests (see p. 17)



BLOCK DIAGRAM OF TYPICAL TWO-TONE RECEIVER TEST SETUP

Test signal power is adjusted for 3 dB increase in level meter reading. DR3 = test signal power – MDS.

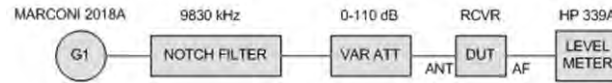
Amplifiers A1, A2 buffer the signal generators G1 and G2 to block RF sneak paths across the combiner. This prevents mixing in the generators' output stages (a cause of IMD).

■9 October 2015

■HF Receiver Testing

■14

Improved RMDR Test Method *using notch filter to improve accuracy*



BLOCK DIAGRAM OF RECIPROCAL MIXING TEST SETUP

- Notch filter (notch at f_0 , depth > 80 dB) between sig. gen. and DUT.
- f_0 = freq. of max. attenuation. Δf = offset.
- DUT tuned to f_0 . Sig. gen. tuned to $f_0 + \Delta f$; input power to raise audio output by 3 dB is noted.
- Notch filter suppresses sig. gen. phase noise at f_0 , thus improving measurement accuracy.
- RMDR = input power – filter passband insertion loss – MDS.

Issues in 2-tone 3rd-order IMD dynamic range (DR3) testing: *subtractive test method*

- ARRL uses subtractive DR3 test method (ITU-R SM.1837 Sec.2).
 - IMD product amplitude is measured at audio output using signal analyzer with 1Hz or 3Hz RBW, to subtract out the noise contribution.
 - The DR3 value obtained via this method is *meaningless* unless RMDR is measured and the result presented alongside DR3.
 - ARRL are now presenting RMDR alongside DR3 in their QST Product Reviews.
- A “100 dB” radio with 85 dB RMDR is *not* a 100 dB radio; it is an *85 dB radio!* To claim otherwise is deceptive advertising.
- If RMDR < DR3, reciprocal mixing noise will mask that “weak one” long before IMD product does.
- In a practical on-air operating environment, artifacts and splatter from distant transmitters will mask weak signals much more often than will IMD in the local receiver.
- This is more an operational and regulatory problem than a technical one.

Issues in 2-tone 3rd-order IMD dynamic range (DR3) testing: *classical test method*

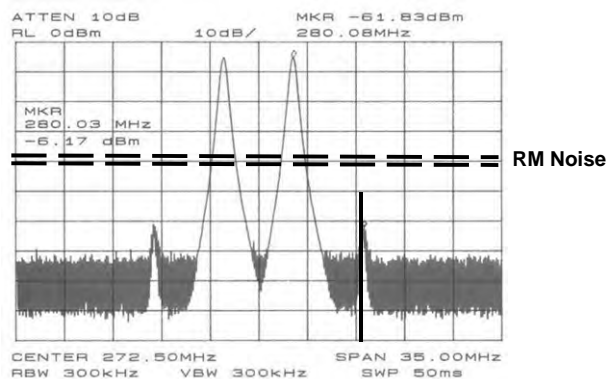
- In the DR3 test method outlined on p. 14, the IMD + noise amplitude is measured at the audio output using an RMS level meter such as the HP 3400 or 339A.
- The test engineer **must** measure RMDR *and* DR3. If RMDR > DR3, the test result is DR3. If RMDR < DR3, we are reading RMDR.
- To check, turn off f_1 and f_2 in turn. If audio output drops by less than 3 dB when either f_1 or f_2 is switched off, the test result is RMDR, not DR3.
- This is acceptable; the test will reveal whether IMD or reciprocal mixing is the receiver's *dominant impairment*.
- In on-air operating, reciprocal mixing (RM) can arise more often than IMD, as only *one* undesired signal will produce RM whereas two are required for IMD to occur.

■9 October 2015

■HF Receiver Testing

■17

Masking of weak signal *when reciprocal mixing exceeds IMD*



IMD Example: $f_1 = 270$ MHz, $f_2 = 275$ MHz.
IMD products at 265 and 280 MHz.
Reciprocal mixing noise masks weak signal.

■9 October 2015

■HF Receiver Testing

■18

Issues in SDR testing: *direct-sampling SDR characterization*

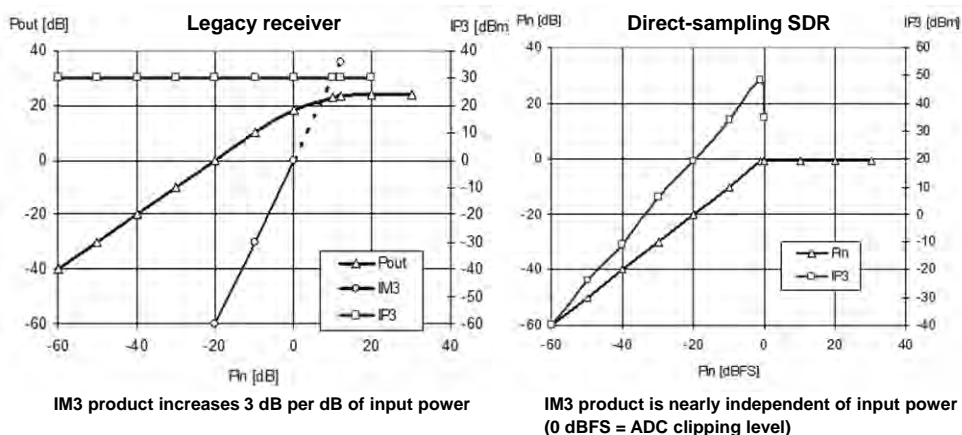
- With the advent of fast, cost-effective ADCs, the direct-sampling SDR has eclipsed its QSD (quadrature-mixer) predecessors.
- This architecture poses new challenges to the test engineer:
- DR3 and RMDR have no relevance as performance metrics.
 - DR3 increases with increasing test-signal power, reaches a peak at ~ -10 dBFS (10 dB below ADC clipping) and then drops rapidly.
- IP3 (3^{rd} -order intercept) is meaningless here, as IMD in an ADC follows a quasi- 1^{st} -order rather than a 3^{rd} -order law.
 - The transfer and IMD curves diverge, and never intersect. In a conventional receiver, IP3 is the convergence point of the transfer and IMD curves.
- As the ADC clock is the only significant phase-noise source, a very-low-noise crystal clock oscillator almost eliminates reciprocal mixing noise.
 - RMDR is so high ($\gg 100$ dB) that even the very best crystal oscillators as test signal sources can degrade the measurement.

■9 October 2015

■HF Receiver Testing

■19

The IP3 Problem in an ADC



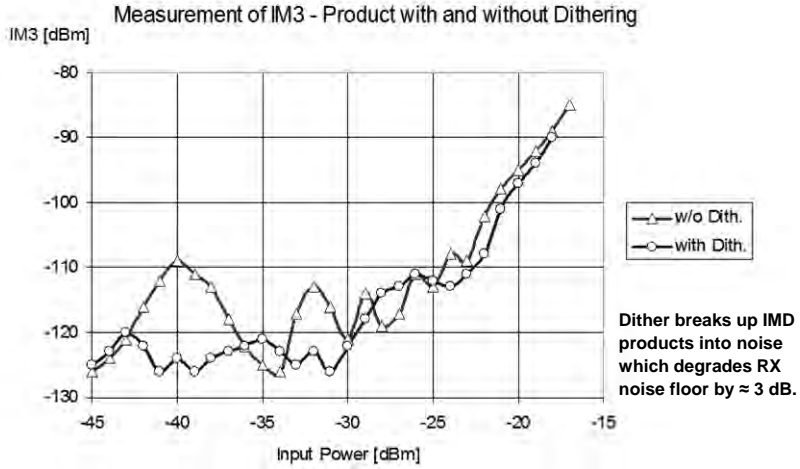
■9 October 2015

■HF Receiver Testing

■20

The effect of dither on IMD

Try this with your old rig!

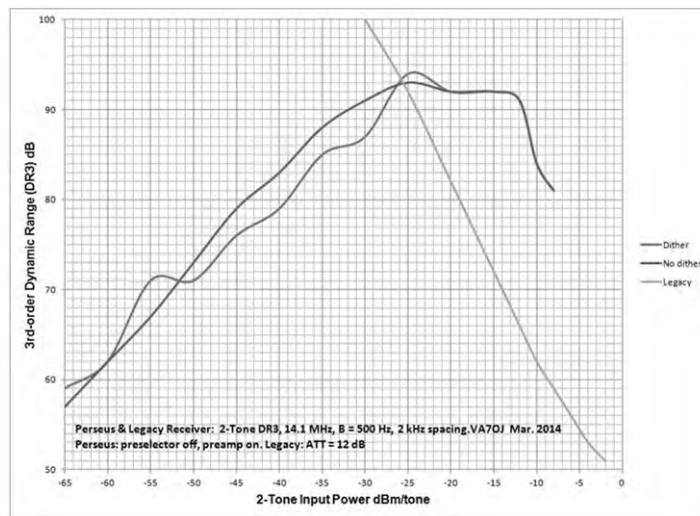


■9 October 2015

■HF Receiver Testing

■21

The DR3 Problem: *Perseus SDR vs. legacy receiver*



■9 October 2015

■HF Receiver Testing

■22

The DR3 Problem:

discussion

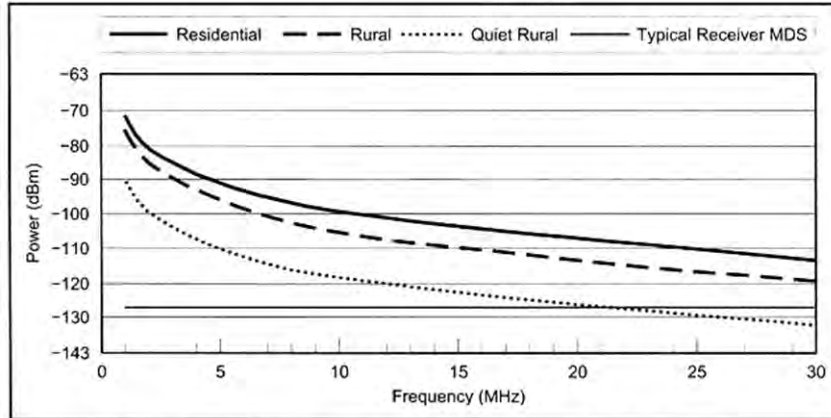
- The chart (see previous page) shows that the DR3 of a direct-sampling receiver is unusable as a predictor of dynamic performance.
- DR3 increases with increasing input power, reaching a “sweet spot” at ≈ -10 dBFS, then falling off rapidly as 0 dBFS (ADC clip level) is approached.
 - By contrast, DR3 of the legacy receiver decreases with increasing input power.
- A new method for specifying receiver IMD is proposed: measure the absolute power of interferers (IMD products and spurs) against 2-tone input power, with the ITU-R P.372 band noise levels for typical urban and rural sites at the frequency of operation as datum lines. We term this IFSS (interference-free signal strength).
 - If the interferer is below the band noise at the user site, the band noise will mask it and it will not be heard.
- The IFSS method allows *comparison* of SDR and legacy receivers.

IFSS IMD Power Measurement

in SDR's and legacy receivers

- We measure the absolute amplitude of each interferer (IMD product or spur) and draw a chart of interferer amplitude vs. per-tone test signal power at a 500 Hz detection or IF bandwidth.
 - The ITU-R P.372-2 band noise levels for typical rural and urban sites (see next page) are shown as datum lines (-103 and -109 dB at 14 MHz, respectively.)
- If the interferer is below the band noise, it can be disregarded.
- The IFSS method eliminates the "sweet spot" problem in DR3 measurements on SDR's, and is valid for SDR and conventional receivers.
- The legacy receiver will often need front-end attenuation to bring its MDS into line with that of the SDR, which is ≈ 10 dB worse as a rule.)
- The IFSS test method allows us to compare the IMD vs. input power performance curves of a direct-sampling SDR and a legacy receiver on a common chart as shown on p. 26.

ITU-R band noise levels (Courtesy ARRL)



Typical noise levels versus frequency for various environments.
(Man-made noise in a 500-Hz bandwidth, from Rec. ITU-R P.372.7, *Radio Noise*)

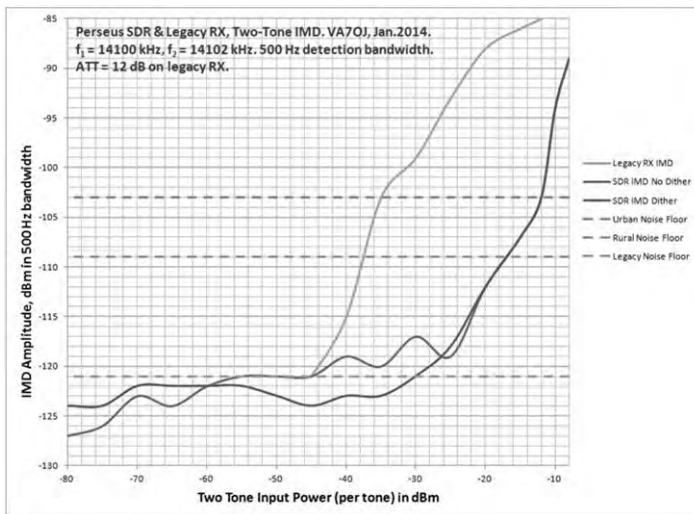
9 October 2015

HF Receiver Testing

25

IMD vs. input power (IFSS): *Direct-sampling SDR vs. legacy receiver*

For the Perseus, the IMD curve is $\approx 1^{\text{st}}$ -order until -25 dBm input level, then rises rapidly to 3^{rd} -order due to IMD in active stages ahead of ADC.



9 October 2015

HF Receiver Testing

26

Measuring dynamic range is easy
but how do we measure absolute interferer levels?

- On a direct-sampling SDR, we can read the observed IMD product and interferer levels directly off the S-meter or spectrum scope.
 - The scope and S-meter level calibration should be checked before taking these readings.
 - A preamp ahead of the ADC will degrade IMD.
- On a legacy receiver, the procedure is more complex.
 - Read recovered audio level of IMD product or interferer on level meter.
 - Next, apply a single-tone test signal to the DUT RF input and adjust input power to obtain same level-meter reading. (AGC must be on.)
- IMD product/interferer levels can also be read off a legacy receiver's *calibrated* signal-strength meter.

Other considerations for HF receiver testing

- The measurement of second-order IMD dynamic range (DR2) is still useful in SDR testing, as 2nd-order mixes in active stages ahead of the ADC can cause HF BCI.
 - Example: 41m BCI on the 40m amateur band in Region 1.
- Image rejection and IF leakage measurements are not applicable to direct-sampling SDR's.
- The **noise-power ratio (NPR)** test is a useful tool for identifying impairments in SDR and conventional receivers.
 - If a complete NPR test set (noise generator and noise receiver) is available, it can be used also for testing 2-port networks (amplifiers, filters etc.)
- <http://www.nsarcc.ca/hf/npr.pdf>

References for further study

1. <http://www.itu.int/rec/R-REC-P.372/en>
2. <http://tinyurl.com/testproc2011> (*ARRL Test Procedure Manual, 2011*)
3. <http://www.nsarca.ca/hf/npr.pdf>
4. http://www.ab4oj.com/test/docs/npr_test.pdf



AMATEUR RADIO EMERGENCY DATA NETWORK AT THE CENTER OF EMERGENCY COMMUNICATIONS PREPAREDNESS

Andre Hansen, K6AH
The AREDN Project (AREDN.org)
2113 Via Monserate
Fallbrook, CA 92028
K6AH@ARRL.net

Abstract

Mesh technology has been around for over ten years. Over the past two years developers on the AREDN™ team have advanced the art by porting Broadband-Hamnet's extremely popular mesh firmware to the Ubiquiti airMAX line of commercial Wireless ISP routers. This has literally changed the complexion of mesh implementations from an experimental, hobby-oriented, novelty into a viable alternative network suitable for restoring some degree of Inter/intra-net connectivity "when all else fails."

More recently, the developers of this software have kicked-off a new project, AREDN, focused on taking this technology to the next level in EMCOMM communications.

This paper begins with an introduction to the AREDN Project and mesh networking and concludes with a roadmap for the Project's future. It dives into implementation techniques and considerations as well as avoidable pitfalls.

Keywords: AREDN, EMCOMM, mesh, BBHN

Introduction

The typical Emcomm message-passing scenario today involves the sender conveying the message to a ham, who transcribes it onto an ICS-213 form. Then the message is spoken over VHF/UHF radio to another ham who writes it down on another ICS-213 form. The form is then delivered to the recipient, who reads it and signs it. The acknowledgement is then conveyed back over the radio to the sending ham who confirms the receipt to the originator.

Emcomm "Customer" expectations aren't being met

Customer expectations differ wildly than this. They expect the continued use of tools with which they are accustomed: email, phone service, chat, and other web-based tools specific to their roles within the organization.

Over \$4B in ham-compatible radios is sold to non-hams each year and most hams wouldn't recognize them to be ham radios. These devices follow the 802.11 standard and operate in several of our microwave bands. They are all around us, and coupled with the privileges our license offers, we should be using this technology to deliver on these customer expectations.

So what is AREDN?

AREDN is an RF network mesh of radio/routers operating under the FCC rules, Part 97 in the ham microwave bands, controlled by hams with a Tech license or higher. It is a high-speed data network with rates of up to 54 Mbps designed to provide a TCP/IP medium when other network infrastructure has failed. While technically capable, it is not intended to be a general Internet access alternative.

AREDN is written for Linux-based WIFI and WISP devices by the AREDN Development Team which also authored the BBHN (Broadband-Hamnet) releases from v1.0.1 to 3.0.1.

AREDN replaces the manufacturer's operating system with the following major components:

1. OpenWRT, an OpenSource wireless routing framework onto which custom applications can be built
2. OLSR (Optimized Link State Routing Protocol), an IP routing protocol optimized for dynamic ad hoc networks
3. Web-based GUI for node configuration
4. Automatic device-specific TCP/IP network configuration based on the device MAC address

The primary objectives of the project are to empower the typical ham to become a deployable part of the network by simply installing the firmware, entering the station's call-sign and an administrative password, and then pointing the node's antenna toward an existing network node.

The secondary objectives are to provide a means to monitor & manage the network and to specify a set of operational standards & services for Emcomm's utilization of the technology.

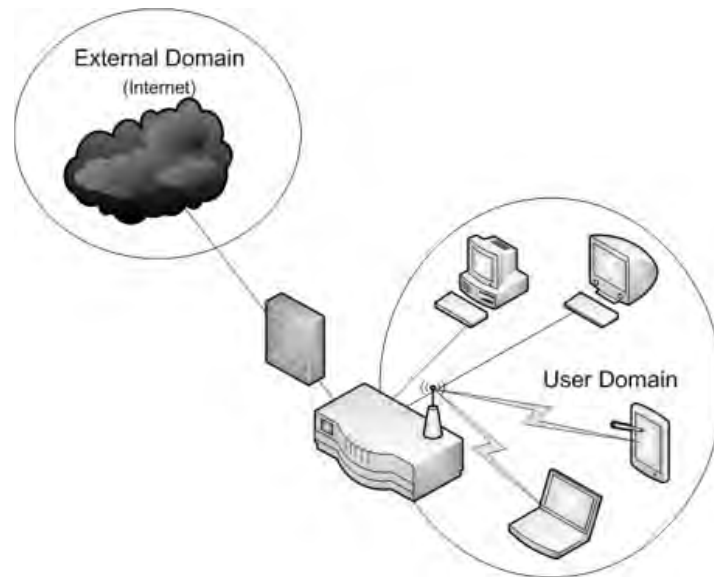
To date this technology has attracted 3 very different user types:

1. Look at the cool things you can do! They're intrigued by the autonomous nature of the network and quickly setup neighborhood networks for gaming, VoIP, etc. They tend to attract other computer types who may be enticed into Ham Radio as a result.
2. Applying it toward a need. These guys weren't looking for it, but see the value in it and apply it toward a specific need, such as Field-Day logging, race support, surveillance cameras, etc.
3. Those who have longed for it... To these guys, the technology is game-changing. They are in the process of exploiting it by building infrastructure around it.

This last type include the Emcomm guys. They are the primary target of this technology and the focus of the AREDN™ Project.

How it works

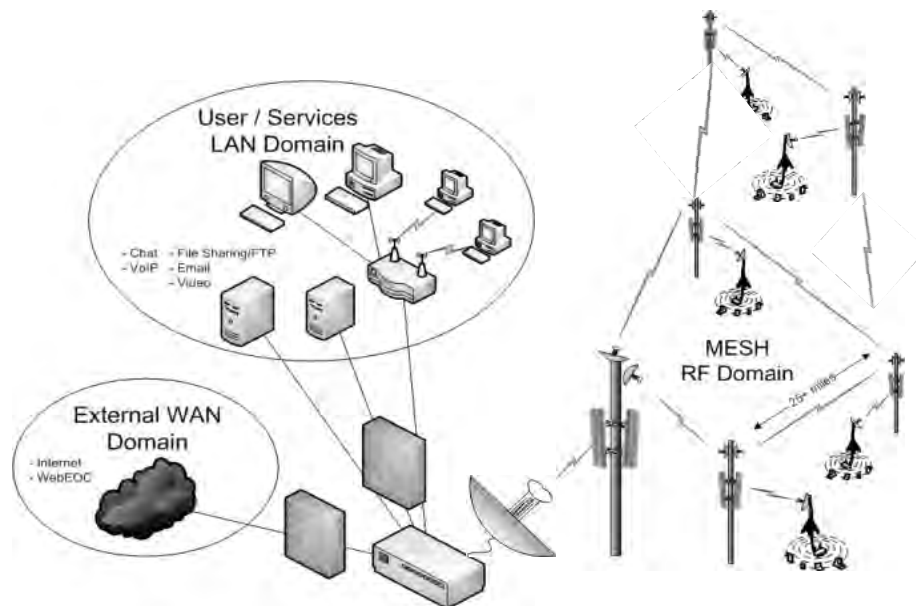
It is easier to understand this technology if we start with how standard WIFI works.



In the diagram above, we see two distinct “domains.” The User domain includes both wired and wireless devices. These are all in the same address space and nothing distinguishes them aside from how they connect to the WIFI router.

The second domain is for the Internet. You will note a firewall protects the User domain from unauthorized access and other threats from the external domain.

How AREDN “repurposes” the device



What were two domains in our Standard WIFI diagram have now become three. We see the familiar external and user domains... although the user domain now contains a WIFI router and new computers which, in this case, deliver services such as email, FTP, VoIP, chat, etc.

The new domain here is an RF mesh network which forms the business end of the AREDN technology.

The Hardware

I'll use this term "mesh" to describe the interconnection of devices, and "nodes" to describe the devices. Nodes are typically comprised of a Linux computer, a software-defined radio operating within a predefined microwave band, and a strip-line amplifier. Some utilize an on-board transverter as a means of reusing an existing device in another band. The computer has at least one Ethernet port, although some have two with an internal hub or switch. All of these components are contained within the single node. The radios contain either internal antenna or one or two antenna ports. These radio receivers are hot... with sensitivities in the range of -95dBm. Power output is in the range of 23 to 28 dBm (200 to 600 mW).

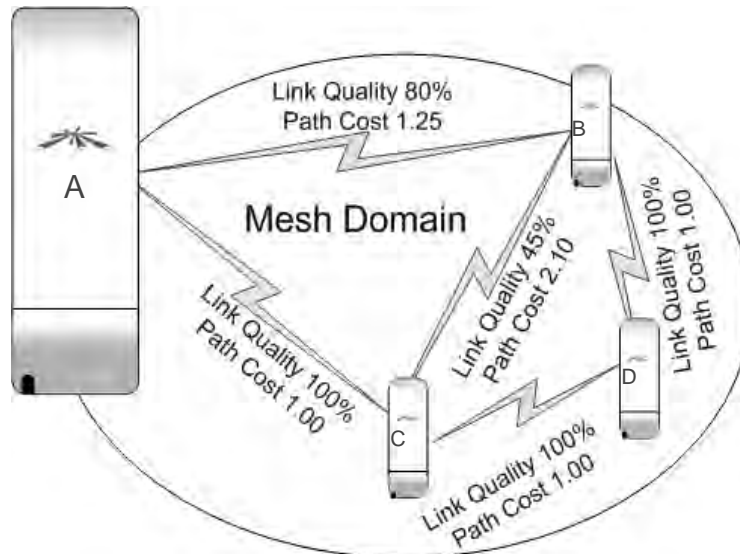
Historically, Ham Radio mesh networks have been built on Linksys WRT54 devices intended for home and office use. These lower-powered (19dBm, 79mW) units have required environmental protection when used outside, and as a result have been difficult to utilize in meshes extending beyond the neighborhood. Over the past 2 years AREDN developers have extended existing mesh technology to environmentally robust, commercially available, Ubiquiti, hardware. The devices supported are currently manufactured by Ubiquiti under their product group "airMAX" and the TP-LINK "Pharos" series. Support of these devices has literally changed the complexion of mesh implementations from an experimental, hobby-oriented, novelty into a viable alternative network suitable for restoring some level of Inter/intra-net connectivity when "all else fails."

The ARDEN software supports these devices on the 900 MHz, 2.4 GHz, 3.4 GHz and 5.8 GHz Amateur Radio bands.



How OLSR Works

Optimized Link State Routing determines the best path for data transmission through the network.



The four devices illustrated above, all Ubiquiti NanoStations, have formed a “mesh.” The route data will take through this network is dependent on the quality of the links between them. Note the link between Node A and Node B. From historical broadcast reception, Node A knows that 80% of the data from Node B is received without error. In addition, Node B knows from historical reception, that data from Node A is received without error 100% of the time... it is said to be 80% reliable based on the following formula, where LQ = Link Quality and PL = % Packet Loss

$$LQ = \frac{PL_{A \rightarrow B}}{PL_{B \rightarrow A}}$$

$$LQ = .8 \times 1$$

$$LQ = 80\%$$

Based on this, it assigns a “cost” (or ETX, estimated transmission) to the A→B link inversely proportional to the Link Quality:

$$\text{Cost (aka ETX)}_{A \rightarrow B} = 1/LQ = 1/.8 = \$1.25$$

All traffic from A to B will take this path, because it is the least expensive route available. To determine the path through multiple nodes, the individual link costs are simply summed.

If the link between A and B were to fail, then Node A would quickly calculate an alternate path. Two are available:

- Path A-C-B at a cost of \$1.00 + \$2.10=\$3.10
- Path A-C-D-B at a cost of \$1.00 + \$1.00 +\$1.00= \$3.00

Node A's routing table is updated with the new optimal path of A-C-D-B. These updates are performed multiple times each second, with routing table propagations through the mesh taking some amount of additional time depending on the size of the mesh.

What the GUI does

The Graphical User Interface is generated using HTTP by the AREDN software running on the embedded Linux computer. The GUI provides access to a variety of administrative and operational functions, such as:

- Checking for traffic / congestion on the channel
- Reporting the current node status and other nodes both directly or indirectly connected
- The basic node configuration settings
- More advanced administrative setting for
 - Port Forwarding / DHCP / Advertised Services
 - Network Address Translations for complex network environments
 - Updating the AREDN firmware
 - Installing useful service packages on the node

K6AH-SleepingIndianEast

WiFi address: 10.48.133.45 / 8
fe80::de9f:dbff:fe30:852d Link

LAN address: 10.132.41.105 / 29
fe80::de9f:dbff:fe31:852d Link

WAN address: none
fe80::de9f:dbff:fe31:852d Link

default gateway: none

Signal/Noise/Ratio: -72 / -95 / 23 dB

firmware version: develop-42-4f18f204 mesh

system time: Set Apr 25 2015 06:00:57 UTC

uptime: 18 days, 18:42

load average: 0.08, 0.07, 0.05

150 KB
90580 KB
= 36764 KB

K6AH-SleepingIndianWest WiFi scan

| Sig | Chan | Enc | SSID or Hostname | MAC | Mode |
|-----|------|-----|------------------|---------------|-----------------|
| -81 | 11 | * | cooMESH1 | 000D97:0817E5 | AP |
| -81 | 11 | * | unknown | 06A0C8:656E98 | AP |
| -81 | 9 | * | Jones | 4C60DE:88D63A | AP |
| -82 | 11 | * | lcmdate | 00A0C8:656E98 | AP |
| -82 | 1 | * | KG9JEI-WEST | 24A43C:92FE67 | BroadbandHamnet |
| -82 | 1 | * | cooMESH1 | 000D97:080C98 | AP |
| -82 | 1 | * | cooMESH1 | 000D97:0617E1 | AP |
| -83 | 1 | * | ATTkUnswgt | D039B3:3FA500 | AP |

K6AH-SleepingIndianWest Basic Setup

Node Name: K6AH-SleepingIndianWest

Node Type: Mesh Node

WiFi

Protocol: Static

IP Address: 10.2.158.200

Netmask: 255.0.0.0

SSID: BroadbandHamnet

Mode: Ad-Hoc

Channel: 1 (2412)

Channel Width: 20 MHz

LAN

LAN Mode: 5 host Direct

IP Address: 10.20.246.65

Netmask: 255.255.255.248

DHCP Server:

DHCP Start: 66

DHCP End: 70

WAN

Protocol: DHCP

DNS 1: 8.8.8.8

DNS 2: 8.8.4.4

Mesh Gateway:

K6AH-SleepingIndianEast signal strength

now: Signal -74, Noise -95, Ratio 21

average: Signal -73, Noise -95, Ratio 22

n = 4/4 max: -73 min: -74 max: -95 min: -95 max: 22 min: 21

How Do I Build an AREDN Network?

Building an AREDN network is not difficult:

1. We encourage one to have a specific objective before you begin. That may be simply to understand the technology, but such an objective should not creep into a production implementation without restarting with that new objective.
2. The next step is to plan and deploy core nodes onto which mesh is formed. These nodes form an initial, primary path for network traffic. Note that they may or may not retain that distinction as the mesh grows.
3. Setup purposeful services that align with your objective and user requirements.
4. Utilize the mesh routinely to ensure it is operational and will be so with needed.

Supported Device Details

Ubiquiti airMAX M-series wireless routers share the following general characteristics:

- They are tower mountable and environmentally robust:
 - Temperature: -40° to +176°F,
 - Humidity: 5 - 95% Condensing
- Many utilize a combination of horizontal and vertically polarized antenna to minimize unwanted interference from on-channel or adjacent WIFI noise. When conditions allow, they also support the combining of these polarizations for increased data throughput—called MIMO (Multi-In, Multi-Out).

Here are a few representative devices and characteristic selection criteria:

- Rocket – A two RF port MIMO node (500mW) that is “plug and play” with a variety of Ubiquiti antenna systems: 90° and 120° Sector antenna, Dual-polarization Verticals, and 30-34dBi Dish antenna. Note that MIMO nodes split the power between the vertical and horizontal domains.
- Bullet – A single RF port high-power (600mW), non-MIMO node with an N-Type female connector suitable for direct connection to many 3rd party antennas. With the right antenna this could easily achieve ranges of 50+ km.
- NanoStation – A fully contained node with an internal 11dBi patch antenna and a 45° coverage pattern.
- airGrid – A larger node available in several size/gain grid-reflector antenna configurations. Designed to be a highly directive, it performs at a range of from 10 to 30 km

All of these devices obtain their operating power delivered over the CAT5 cabling (PoE or Power over Ethernet). They have a broad DC input voltage specification to accommodate a wide range of cable lengths which may be required for tower applications: 10.5 to 24VDC at the CAT5 connector.

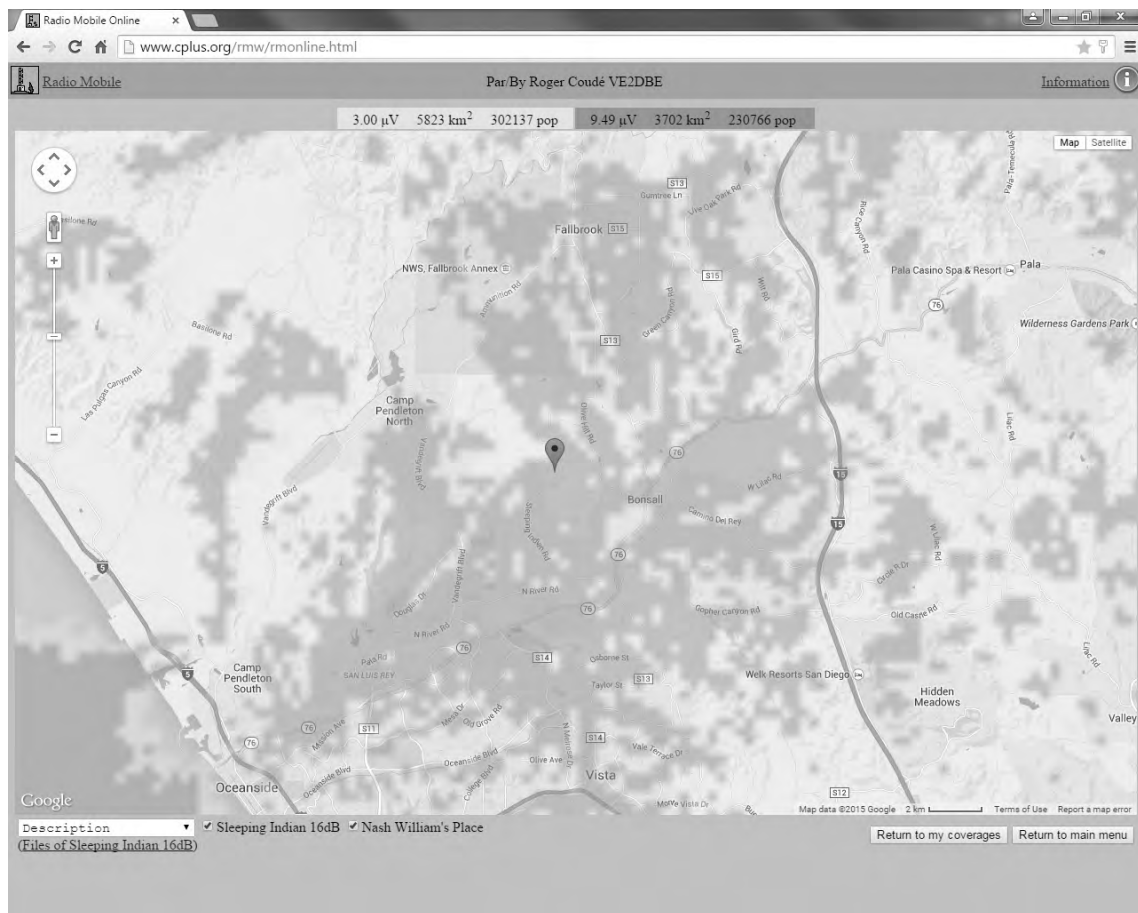
In planning to deploy the core nodes it is advisable to use propagation prediction software such as Radio Mobile to avoid the hassle and expense of experimentation. I will diverge for a bit for the benefit of those unfamiliar with this type of software.

Radio Mobile is a free propagation simulation system. It utilizes data from the Space Shuttle Radar Terrain Mapping Mission (SRTM) resulting in elevation contours which have then been overlaid with satellite imaging and road maps. It further utilizes the Longley-Rice radio propagation prediction method, which computes the attenuation of radio signals using an “irregular terrain model”... a technique that has been successfully used in commercial radio coverage planning since the 1960’s.

It is available both as a software download and a Web-based tool. While the download results in a more flexible tool, its installation is not for the faint-of-heart. I would advise that, if you do not consider yourself a computer expert, then the Web-based tool will more than adequately meet your needs. The English language portal is at: <http://www.cplus.org/rmw/english1.html>

Sufficient use of this tool to explore the variables of band, node-model receiver sensitivity, and node-model power output, will result in the required antenna gain in either point-to-point (PtP) or point-to-multi-point (PtMP) topologies.

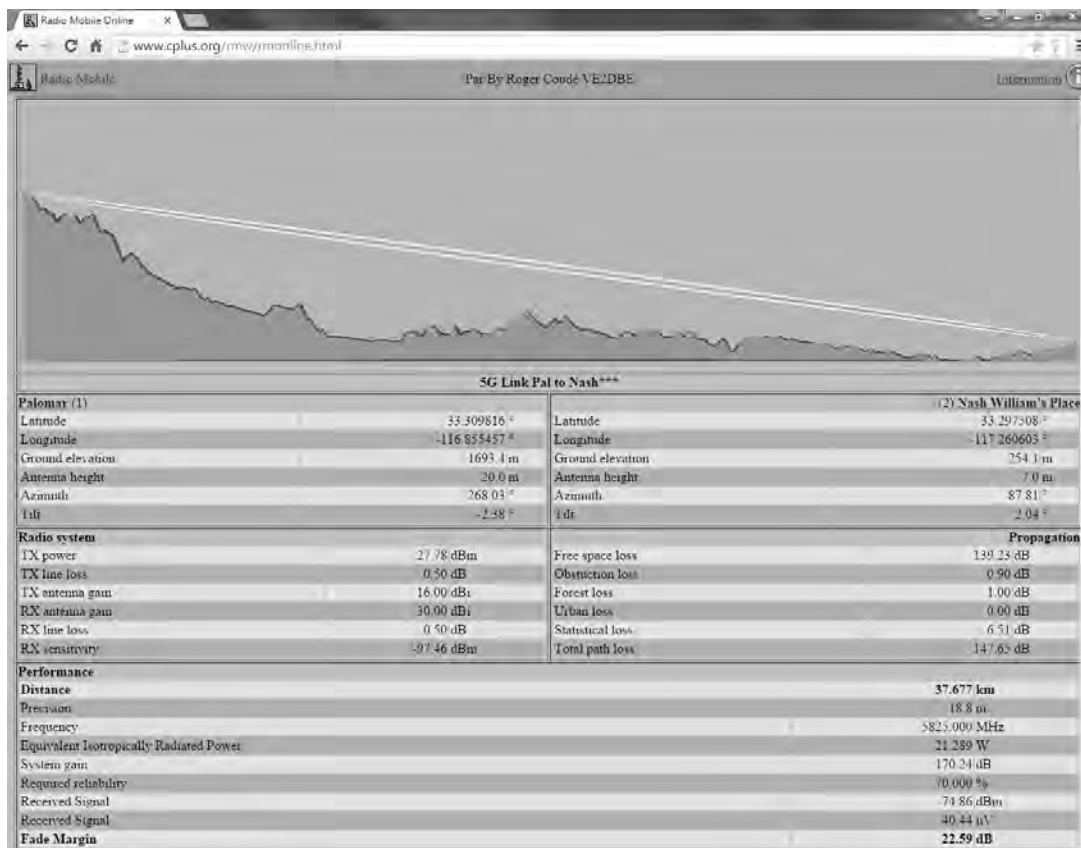
Here are a screenshot from a typical Radio Mobile analysis:



In my planning for a mesh in northern San Diego County, I secured a location in the Bonsall area. The marker locates the potential node(s) under review... in this case a pair of 2.4 GHz Ubiquiti Rocket nodes with 120° sector antenna, one pointed in the general southwest direction and other to the northeast. Based on the specific parameters I entered for these nodes the green-shaded areas will hear the node under review at approximately 9.5µV (-87.5dBm, 7.5dB above the receive sensitivity) and the yellow area at about 3µV (-97dBm, 2dB below the limit of the receiver sensitivity).

I was interested in connecting this cluster of nodes to a high-ground “backbone” node recently placed near Mt Palomar, a 5000’+ peak directly to the east.

The analysis proves out the viability of the prospective link with a more than ample fade margin of 26dB. The author normally considers a margin of 15dB or more as adequate in most instances. Whether this is achievable is highly dependent on the RF environment the node is being placed. Under ideal circumstances you want to take advantage of the full receiver sensitivity (approx. -95dBm for most Ubiquiti devices). But this may not be possible if there is in-band competing activity from other nodes, or other RF sources. For any given situation, it is the signal-to-noise ratio (SNR) that is important, not the signal strength above the receiver’s sensitivity. SNR is also expressed in dB. For example, if the noise floor at a given location is -85dB, then you would need a received signal strength of -70dBm in order to achieve an SNR of 15dB. Remember that Radio Mobile, or any other predictive software is not knowledgeable of the noise environment you will encounter. So uncertainty will remain until you visit the prospective site and confirm this variable.



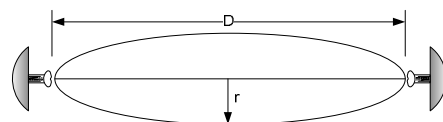
Power Density

As hams we generally don't concern ourselves with too much with power density. Common modulation techniques such as CW, SSB, AM, FM are relatively narrow, but that's not the case with 802.11b which uses direct sequence spread spectrum (complementary coded keying - CCK) or 802.11a/g which utilize orthogonal frequency division multiplexing (OFDM). The AREDN software allows user-selectable bandwidths of 5, 10, and 20 MHz.

So why would a user want to reduce the bandwidth... and the corresponding maximum throughput of the link? The answer is, to increase the signal-to-noise ratio. Each halving of the bandwidth will improve the SNR by 3 dB. Therefore, going from 20MHz to 5MHz has a 6dB improvement. This can easily represent the difference between a reliable link and problematic one.

Fresnel Zones

One more technical concept you need to be cognizant of is Fresnel (pronounced "fren-l") Zones. They are imaginary oblong lines which emanate from one node to another resembling an elongated cigar. The primary cigar represents an area within which interfering objects will cause destructive signal reflections at the receiving end. This is also referred to as "multi-pathing." The formula for computing this area is closely approximated by the following simplified formula:


$$r \text{ (in feet)} = 36 \sqrt{\frac{D \text{ (in miles)}}{4f \text{ (in GHz)}}$$

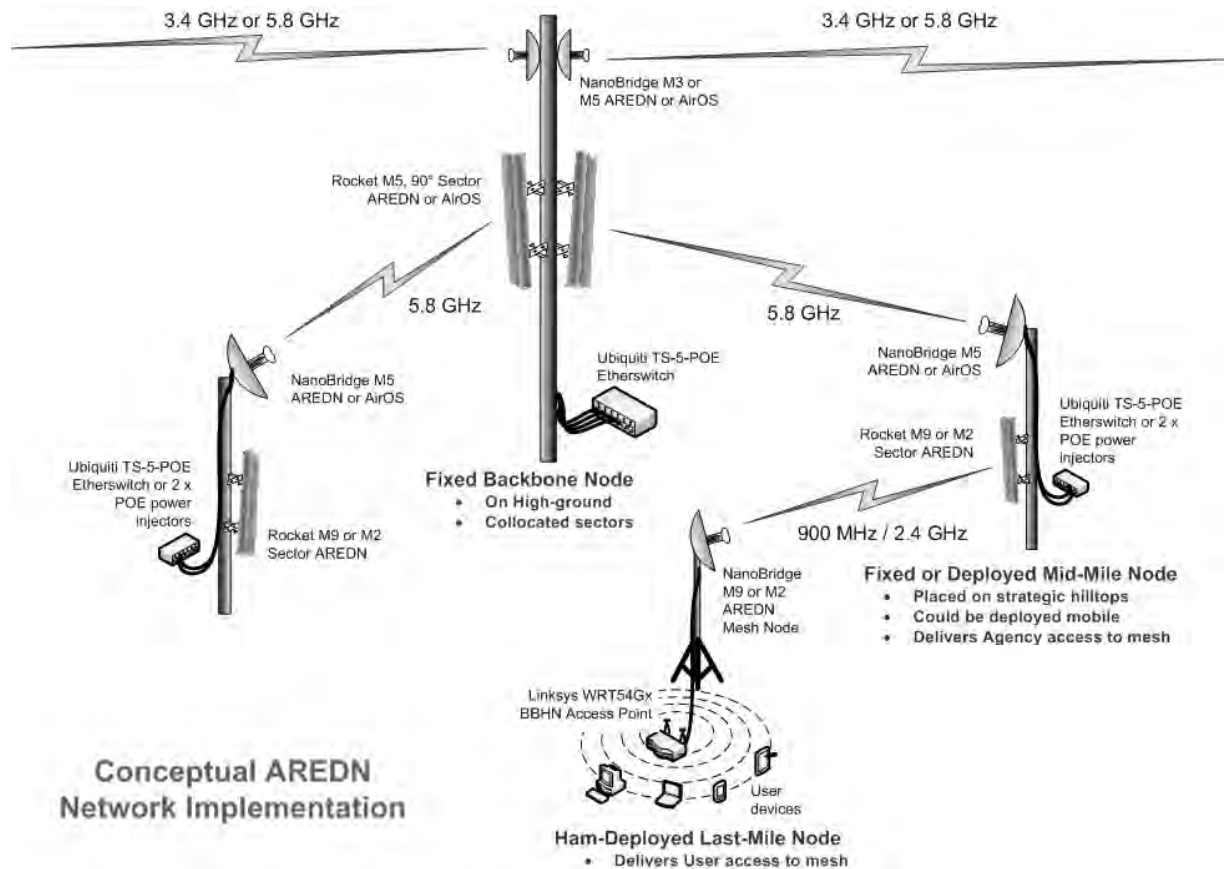
This zone should be clear of all obstructions, including vegetation. If your modeling exercise is not playing out in the real world, the likely culprit is an obstruction in the Fresnel Zone. Notably, the 900 MHz band is tolerant of some vegetation in this zone.

For those less mathematically adept, here are some examples:

| | |
|---------------------------|---------------------------|
| 900 MHz at 10 miles = 60' | 900 MHz at 20 miles = 85' |
| 2.4 GHz at 10 miles = 37' | 2.4 GHz at 20 miles = 52' |
| 3.4 GHz at 10 miles = 31' | 3.4 GHz at 20 miles = 44' |
| 5.8 GHz at 10 miles = 24' | 5.8 GHz at 20 miles = 33' |

Conceptual AREDN Network Implementation

Below is a conceptual layout of an AREDN network's core nodes:



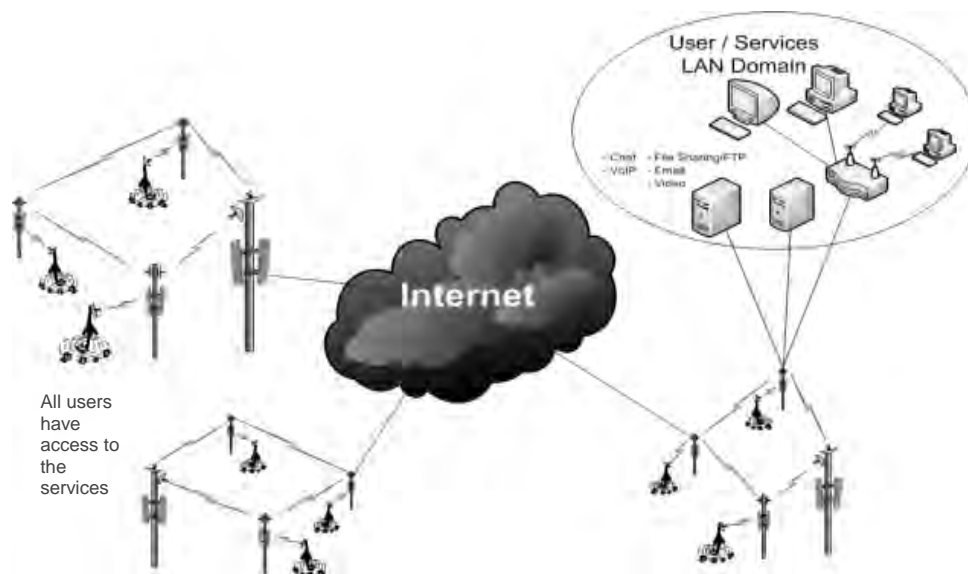
While the term “mesh networking” implies a peer relationship between nodes, a structured core of nodes is defined to establish an initial, predictable path for the network. Three node types are illustrated.

- **Ham-Deployed Last Mile Node:** As the name indicates, this node is carried by the Ham deployed to a served-agency location requiring the pre-established data services. The “Go-Kit” is comprised of a 2.4 GHz node, or for longer, outlying areas a 900 MHz node with a high-gain antenna pointed at a Mid-Mile Node. The kit also contains a WIFI router to provide network access for the local devices on site. Batteries or a generator powers these devices.
- **Mid-Mile Node:** This node is either fixed or vehicle deployed as necessary and per the specific network’s design. It forms a “reachable” collection point for surrounding Last Mile Nodes and has the required higher-gain antenna to reach a higher-ground based Backbone Node. Sector antenna(s) allow broad downstream accessibility.

- **Fixed Backbone Node:** These nodes are permanent installations that extend the mesh to the extreme corners of the planned coverage area. For reliability they operate on the least congested band and are optimized for data throughput. Again here, sector antenna(s) are utilized to maximize their downstream accessibility to Mid-Mile Nodes.

Deployment Challenges

One of the most challenging aspects of a mesh implementation is inter-connecting “mesh islands” that have formed in the more-easily meshed areas. Without a completed mesh network, it is difficult to justify the expense/effort of building out network services (email, Voice-over-IP (VoIP) telephony, web-based utilities, etc.) which are needed to demonstrate the network to prospective EMCOMM clients. It may also be difficult to justify the expense of acquiring strategic high-ground properties necessary to connect the mesh-islands.



The interim solution AREDN has provided is based on Internet tunneling. This involves setting up an encrypted tunnel between one tunnel server-node and one node in each of the other mesh-islands. This has the effect of connecting all participating mesh-islands together in the same network. In doing so, you can gain the benefits of having completed the network and, at the same time justify the build-out of IP-based services for the users and demonstrate the utility to prospective customers.

While tunneling is an effective way to gain that critical mass, it is a poor strategy for EMCOMM deployment and should only be used as a temporary means of achieving a specific goal. Tunnels will likely not be functional in a real disaster.

A Roadmap for the Near Term

The AREDN Development Team recognizes that the following important improvements are warranted and is commitment to the advancement of this technology.

A More Advanced GUI – More advanced users have taken to other network devices in deploying more sophisticated network topologies. We are committed to providing an optional, more advanced GUI to allow for the configuration of these progressive designs. The effort will preserve the auto-configuration features for the non-network savvy Hams.

Network Management with SNMP – Utilizing the Simple Network Management Protocol, the AREDN network should be manageable using any number of standard tools in use today. Customized MIBs (Management Information Bases) will be developed specific to this technology.

These tools will allow visualization of the mesh against a map background, see the network throughput at a node-interface level, locate choke points within the network that would benefit from improved RF quality and bandwidth, and be alerted to segment outages.

Quality of Service (QoS) – Critical traffic needs priority. EMCOMM data during a disaster must not be hampered by casual traffic. Therefore, some form of Quality of Service is warranted. We are not certain how this would best be implemented, but will likely require some advanced configuration or certification at the user-level.

Operations outside of the ISM bands – The available Ham band is broader than the Ubiquiti and TP Link devices support in their manufactured form. There would be great value in moving the device off the ISM-portion of the band in terms of reduced interference and higher resultant SNR. We are exploring this and believe this will be possible.

Preserving the 3.4 GHz band – Having recently released software supporting devices in the 3.4 GHz band, the team is just coming to realize the huge asset this band represents. This band has no commercial allocation in the US and is considered to be in jeopardy of loss to commercial interests. With the exception of some military radar, there is little interference, making it the quietest of alternatives. A Ham presence, particularly in the EMCOMM space, could prove useful in retaining it for Ham purposes.

AREDN Team Developed Deployments – The number of network deployments utilizing this technology are too numerous to list, however it is fair to say that groups in most major US cities are, at a minimum, exploring this technology as well as major cities in Canada and Europe.

Where 2 years ago I was begging for speaking opportunities, they now approach me. I am certain we are past the tipping point.

How One Gets Involved – Most local deployments start at a grass-roots level. It is an excellent way to span the gap between radio and computer technologies... and, if approached correctly, can attract a new generation to Ham Radio. Larger implementations deserve a network specialist, so I encourage you to find one early. Hams tend to become overwhelmed with the networking elements, so having someone to offload that worry lets Hams worry more about the radio aspects of this technology than the data.

There is a fabulous getting-started primer entitled “Wireless Networking in the Developing World” which I encourage anyone interested in this technology to read. It is available for free as a PDF download at: www.wndw.net.

Conclusion

There are a variety of mesh network systems today. AREDN is unique in that it operates under Part 97 under the authorizations inherent in our Amateur license grant. It is easy to configure and is deployable by typical hams to served agencies without any knowledge of data networking or the design of the mesh to which a node is being connected. It can be used to provide a variety of IP-based services or to restore failed intranet-based agency services.

The AREDN Project team provides support via its website at www.aredn.org to Emcomm groups wishing to deploy this technology.

The AREDN logo is a copyright of Randy Smith, WU2S, and is used with permission.
Broadband-Hamnet is a trademark of the Broadband-Hamnet, Inc.
airMAX, NanoStation, airGrid, Bullet, and Rocket are trademarks of Ubiquiti Networks, Inc.
TP-LINK and Pharos are trademarks of TP-LINK Technologies, Co., Ltd.

The AREDN software is distributed and licensed for use under the Free Software Foundation’s General Public License, GPLv3 license. A copy of that license and source is available at the project’s web site: <http://www.aredn.org>.

Feher Modulation 16 QAM

Patrick Jungwirth, PhD
US Army RDECOM
Aberdeen Proving Ground, MD 21005

Abstract

We present simulations of conventional quadrature amplitude modulation (QAM) and Feher-QAM to estimate the bandwidth improvement for Feher-QAM. We show more than a 10% improvement (reduction) in bandwidth for Feher-QAM over conventional QAM. We also show the power spectral density for Feher-QAM has a much faster convergence than conventional QAM.

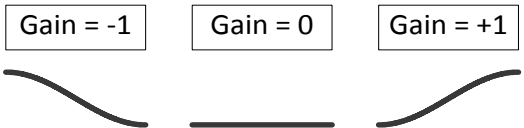
Conventional digital modulation techniques are limited by embedded rectangular windowing functions. Some more advanced modulation techniques utilize raised cosine windowing functions (filters) to improve sidelobes. Feher modulation uses a half cycle raised cosine waveform to reduce bandwidth and improve sidelobe attenuation. Feher modulation offers the equivalent power spectral density convergence of a raised cosine windowing function with twice the width (half the bandwidth). All symbol transitions in Feher modulation are smooth and occur at zero slope points. The smooth, zero slope transitions help improve intersymbol interference, and reduce timing jitter problems.

Key words: Feher modulation, QAM, Feher-QAM

1. Introduction

We present a short introduction to Feher, half cycle raised cosine, modulation. Feher modulation consists of half cycle raised cosine functions concatenated together to form a smooth function. For the symbol sequence $\{0, 1\}$ a step change occurs when transitioning from 0 to 1 in conventional modulation. In Feher modulation, the symbol transition occurs over a full symbol time. The sequence $\{0, 1\}$ is mapped to a positive going half cycle raised cosine waveform, \nearrow , with gain = +1 in (1.1). The sequence $\{1, 0\}$ is mapped to a negative going half cycle raised cosine waveform, \searrow , with gain = -1 in (1.1). The sequences $\{0, 0\}$ and $\{1, 1\}$ are mapped to a zero slope line segment, $—$, as shown by gain = 0 in (1.1).

Half Cycle Raised Cosine
(basis function)



The diagram shows three waveforms corresponding to different gain values. Above each waveform is a box containing the gain value: Gain = -1, Gain = 0, and Gain = +1. The Gain = -1 waveform is a negative-going half-cycle raised cosine wave. The Gain = 0 waveform is a horizontal line segment. The Gain = +1 waveform is a positive-going half-cycle raised cosine wave.

$$(1.1)$$

The mapping of serial data $\{0, 1, 1, 1, 0, 0, 1\}$ to Feher modulation is illustrated in Figure 1.1. The initial symbol (condition) is assumed to be 0 . The 0^* to 0 transition is mapped to a zero slope line segment (where 0^* is the assumed initial symbol). The next symbol transition is $0 \rightarrow 1$ which is mapped to a positive going half cycle raised cosine waveform, \nearrow . The $1 \rightarrow 1$ transition is mapped to a zero slope line segment, $—$. The $1 \rightarrow 1$, $1 \rightarrow 0$, $0 \rightarrow 0$, and $0 \rightarrow 1$ mappings are also shown in Figure 1.1. As illustrated in Figure 1.1

Feher modulation is a smooth function with zero slope points occurring at the serial data symbol transition points (dashed vertical gray lines). Figure 1.1 also shows that each half cycle raised cosine waveform (gain = -1, 0, and +1) occurs over one serial data symbol time.

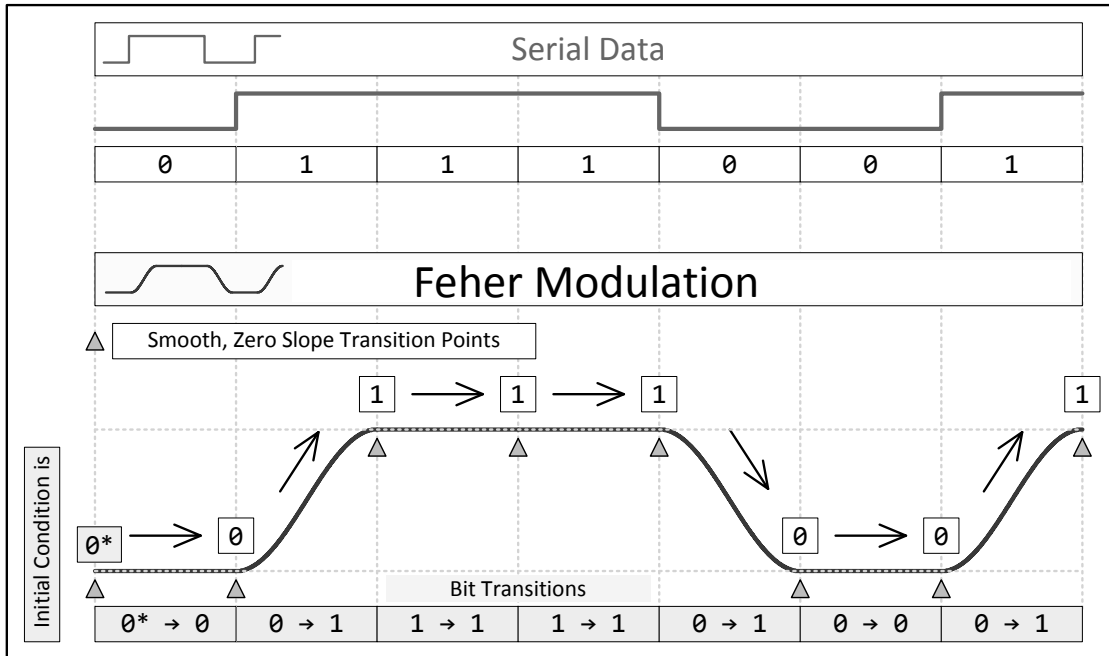


Figure 1.1. Feher Half Cycle Raised Cosine Modulation

2.0 Windowing Functions

Equation (2.1) introduces the unit step function. The unit step function is used to create the rectangular windowing function in (2.2). Equations (2.2) through (2.5) introduce a number of windowing functions. When data is sampled over a finite length of time, the resulting function is equivalent to a rectangular windowing function times an infinitely long function. The rectangular windowing function has slow (poor), $dB(f) = 20 \log \left(\frac{\sin(f)}{f} \right)$, power spectral density convergence as illustrated in Figure 2.1. To improve the convergence of sampled data, a raised cosine windowing function is used to reduce the effects of the rectangular windowing function.

For the serial data stream in Figure 1.1, the step changes embed rectangular windowing functions in the serial data's power spectral density [1-5]. From a practical point of view, Feher modulation minimizes the slope when transitioning from serial data symbol, n , to the next serial data symbol, $n+1$. Full cycle raised cosine modulation, and overlapped raised cosine modulation are described in [6-8].

The half cycle raised cosine windowing function (2.5) has the same power spectral density convergence as a raised cosine window with twice the width (2.4). In section 4, simulations are used to determine the bandwidths for conventional quadrature amplitude modulation, and Feher-QAM. We will show in sections 4 and 5 that the half cycle raised cosine windowing function results in more than a 10 % reduction in bandwidth for 16 quadrature amplitude modulation.

Unit step function, $u(t)$



$$(2.1)$$

Rectangular windowing function (width = 2)

$$w_{rec}(t) = u(t + 1) - u(t - 1)$$



$$(2.2)$$

Raised Cosine windowing function (width = 2)

$$w_{rc}(t) = \begin{cases} \frac{1+\cos 2\pi t}{2} & -1 \leq t < 1 \\ 0 & \text{otherwise} \end{cases}$$



$$(2.3)$$

Raised Cosine windowing function (width = 4)

$$w_{rc}(t) = \begin{cases} \frac{1+\cos \pi t}{2} & -2 \leq t < 2 \\ 0 & \text{otherwise} \end{cases}$$



$$(2.4)$$

Half Cycle Raised Cosine windowing function

$$w_{hc}(t) = \begin{cases} \frac{1+\cos(\pi t - \frac{\pi}{2})}{2} & -1 \leq t < 1 \\ 0 & \text{otherwise} \end{cases}$$



$$(2.5)$$

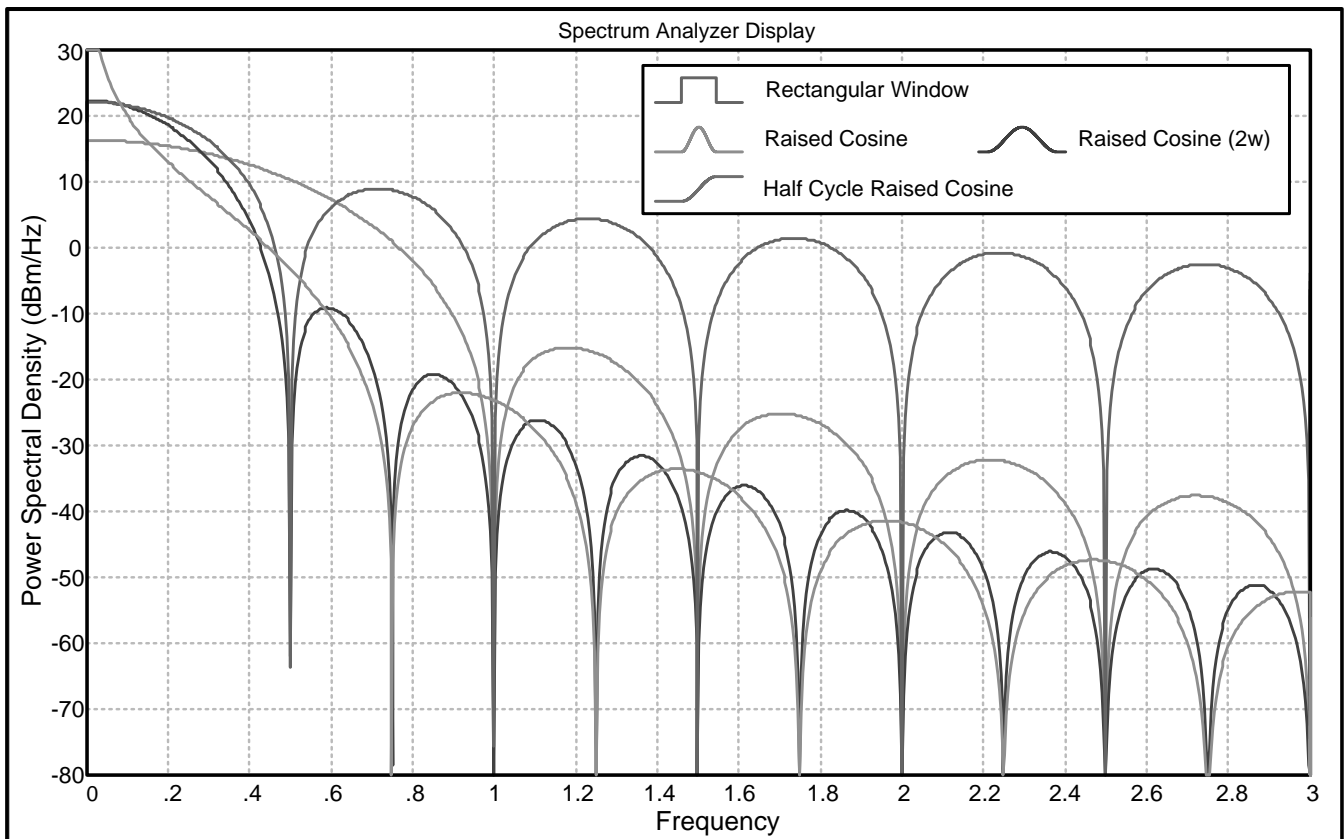


Figure 2.1. Windowing Functions

3.0 Quadrature Amplitude Modulation (QAM) Background

Quadrature amplitude modulation(QAM) $I(t)$ and $Q(t)$ values are found in the diagrams in Figure 3.1 and Figure 3.2. (3.1)

$$s(t) = I(t)\cos(2\pi f_c t) - Q(t)\sin(2\pi f_c t) \quad f_c = \text{carrier frequency in Hz}$$

Equation (3.1) provides the general definition for quadrature amplitude modulation. Cosine and sine terms are multiplied by $I(t)$ and $Q(t)$ signals respectively. The term $I(t)$ is in-phase with the cosine carrier and $Q(t)$, the quadrature phase term, is 90° ($\frac{\pi}{2}$ radians) out of phase with the cosine carrier. Figure 3.1 presents the constellation diagram for 4 level QAM which is equivalent to 4 level phase modulation. We see the points on the constellation diagram all have a radius of 1 with phase angles at $\pm 45^\circ$ and $\pm 135^\circ$ ($\frac{\pm\pi}{4}$ and $\frac{\pm 3\pi}{4}$ radians). The angle sum identity shows 4 level QAM reduces to phase modulation. This is a special case; in general the I and Q terms create both amplitude and phase modulation.

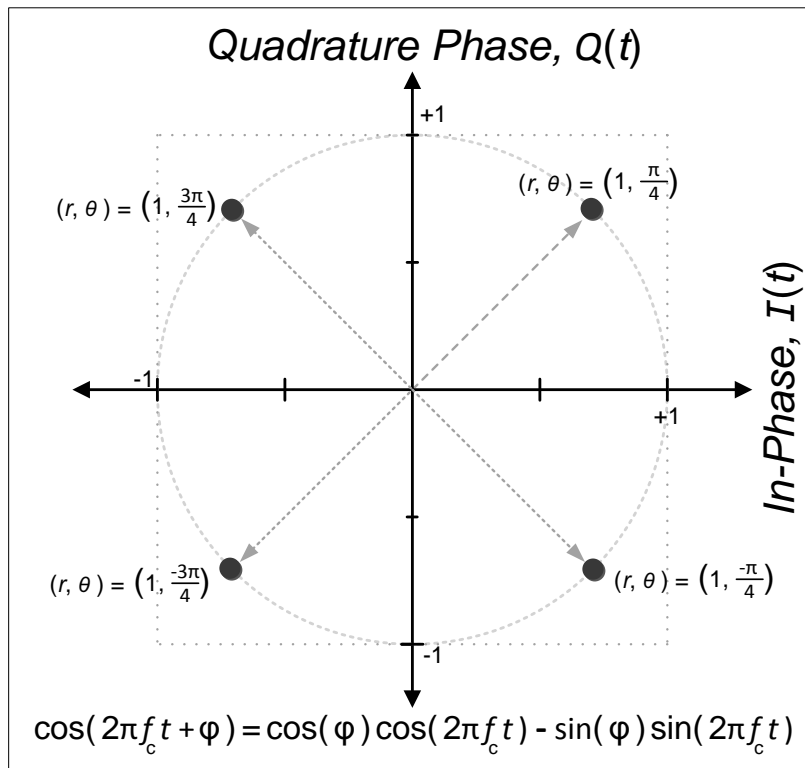


Figure 3.1. 4 Level QAM or 4 Level Phase Modulation

$$\text{Message} = \text{“DSP”} = 0x44; 0x53; 0x50 \text{ (where } 0x\#\# \text{ indicates a hexadecimal number)} \quad \{3.1\}$$

We present an example message = “DSP” to show how ASCII characters are converted to 16 level quadrature amplitude modulation. The message “DSP” in {3.1}, converted to ASCII code gives “D” = 0x44, “S” = 0x53, and “P” = 0x50. For 16 QAM in Figure 3.2, there are 16 symbols {0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF}. Each group of 4 bits from the

message “DSP” is converted to a (I, Q) vector representation for 16 QAM modulation. Figure 3.2 shows the 4 bit code to 16 QAM look-up table. For “D”=0x44, the 4 bit groups are 0x4; 0x4, the (I, Q) vector for 0x4 is (0.75, 0.25) volts. Equation (3.2) shows the (I, Q) vector values for the message in {3.1}. The QAM signal, $s(t)$ as shown in (3.1), is calculated from the n -th (I, Q) vector and the cosine and sine carrier terms.

Figure 3.3 shows a 16 level QAM modulator block diagram and simulation. Figure 3.3 clearly shows step functions present in 16 QAM. The embedded step functions (rectangular windowing functions) result in a $\text{sinc}(x)/x$ power spectral density function as shown in Figure 3.3.

$$\begin{aligned}
 &\text{Message} = \text{“DSP”} = 0x44; 0x53; 0x50 \\
 &\text{“D”} = 0x4; \quad 0x4 \quad \text{“S”} = 0x5; \quad 0x3 \quad \text{“P”} = 0x5; \quad 0x0 \quad (3.2) \\
 (I, Q)_{(n)} = &(+0.75, +0.25), (+0.75, +0.25); (+0.25, +0.25), (-0.25, +0.75); (-0.25, +0.25), (+0.75, +0.75)
 \end{aligned}$$

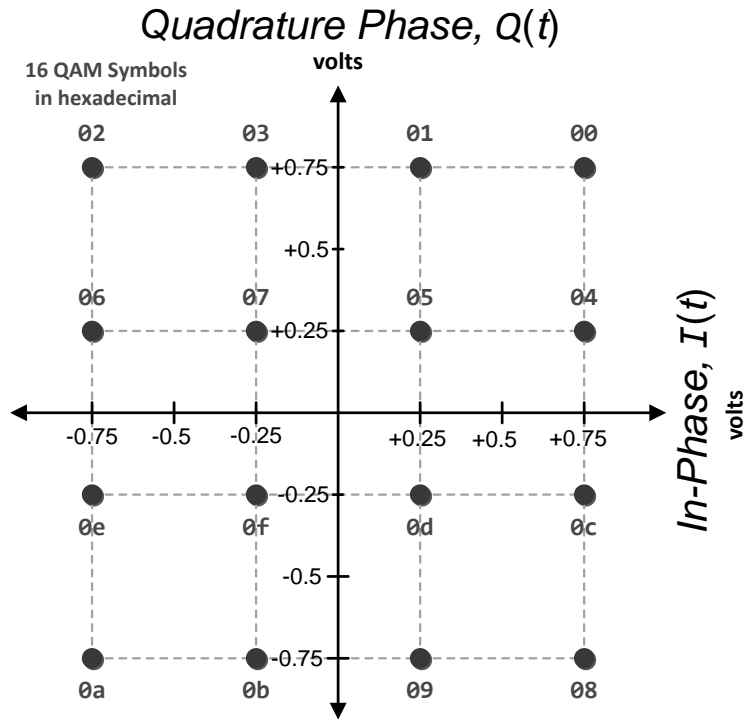


Figure 3.2. 16 QAM Constellation Diagram

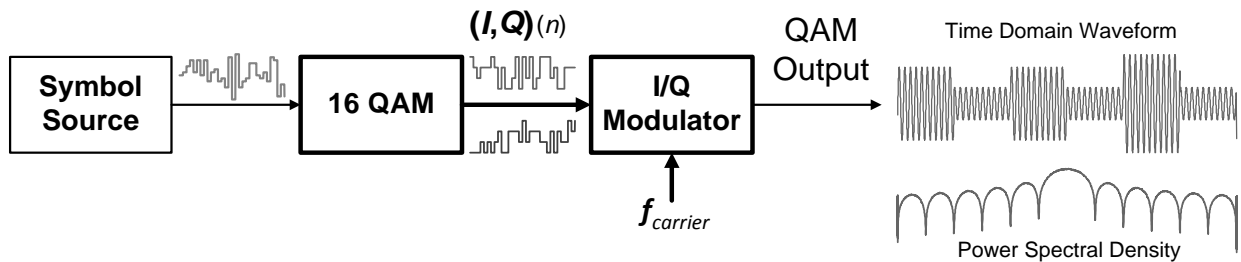


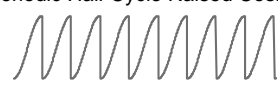
Figure 3.3. 16 QAM Modulator Block Diagram

4.0 Feher-Quadrature Amplitude Modulation (QAM)

Feher quadrature amplitude modulation (QAM) is similar to the binary example in Figure 1.1. Serial binary data consists of two amplitude values +1 and -1 (or 1 and 0). 16 QAM consists of 4 amplitude values for I and 4 amplitude values for Q . As illustrated in Figure 3.2, the 4 amplitude values are -0.75, -0.25, +0.25, and +0.75. The half cycle raised cosine functions are gain scaled by (4.1) to connect the (I, Q) QAM symbols together. For the half cycle raised cosine waveform, the initial value, $DC_Offset(n)$ in (4.2), is the final value from the previous symbol. $H_{RC}(t)$ in (4.3) generates unit amplitude, half cycle raised cosine waveforms. Feher QAM in (4.4) is a unit half cycle raised cosine, $H_{RC}(t)$ in (4.3), gain scaled by (4.1), plus the final value from the previous (I, Q) vector in (4.2).

$$\mathbf{Gain}(n) = (I, Q)_{(n)} - (I, Q)_{(n-1)} \quad (4.1)$$

$$\mathbf{DC_Offset}(n) = (I, Q)_{(n-1)} \quad (4.2)$$

$$H_{RC}(t) = \frac{1}{2} \left[\cos(2\pi f_{sym} \sigma) + 1 \right] \quad \text{Periodic Half Cycle Raised Cosine} \quad (4.3)$$


$\sigma(t) = \text{锯齿波}$ Where $\sigma(t)$ is the sawtooth function

$$\mathbf{Feher_QAM}(t) = H_{RC}(t) \cdot \mathbf{Gain}(n) + \mathbf{DC_Offset}(n) \quad (4.4)$$

Figure 4.1 shows a block diagram implementing Feher modulation algorithm for 16 QAM. Vector operations are shown as **thick lines**. Scalars are shown by thin lines. The most complicated part of Feher modulation algorithm is the unit half cycle raised cosine generator, $H_{RC}(t)$ in (4.3). The rest of the operations are vector addition and scalar multiplication. A look-up table (Figure 3.2) converts 4 bit numbers to 16 QAM symbols.

Conventional QAM and Feher QAM are compared in Figure 4.2. The half cycle raised cosine waveforms form smooth curves from $(I, Q)_{(n-1)}$ to $(I, Q)_{(n)}$. The Feher QAM waveforms are smooth functions without step changes. At each conventional QAM symbol transition, Feher QAM has a zero slope. Each half cycle raised cosine requires a full conventional QAM symbol time to change state. Figure 4.1 shows a Feher 16 QAM modulated waveform. It is a smooth function without any step changes.

Figure 4.3 and Figure 4.4 compare simulations of conventional 16 QAM to Feher 16 QAM. Figure 4.4 shows an expanded scale highlighting the main lobes. Conventional QAM has a $\sin(x)/x$ power spectral density function with a slow convergence. Feher QAM has a narrower main lobe with a much faster convergence (cosine-like windowing function). Section 5 compares bandwidth and convergence for conventional QAM and Feher-QAM.

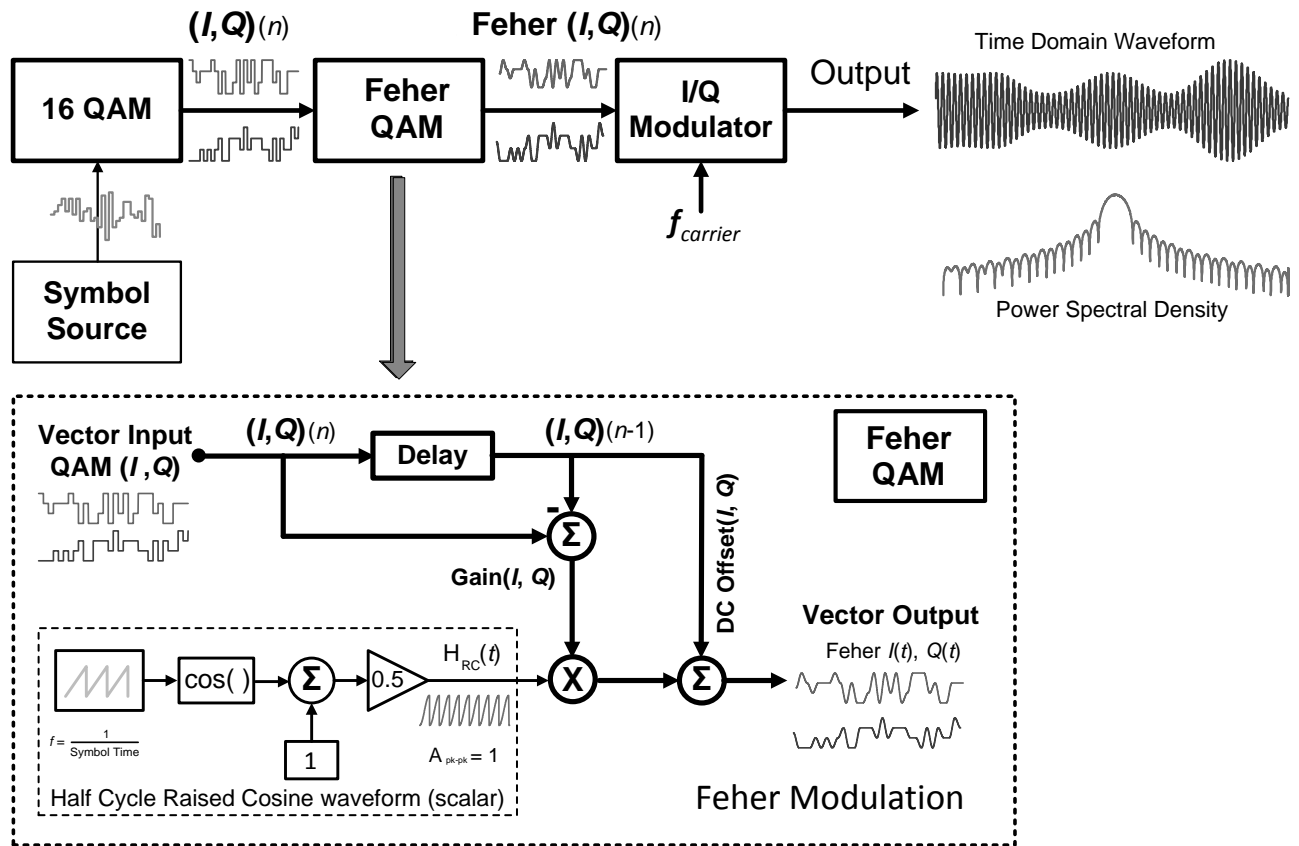


Figure 4.1. Feher-QAM Block Diagram.

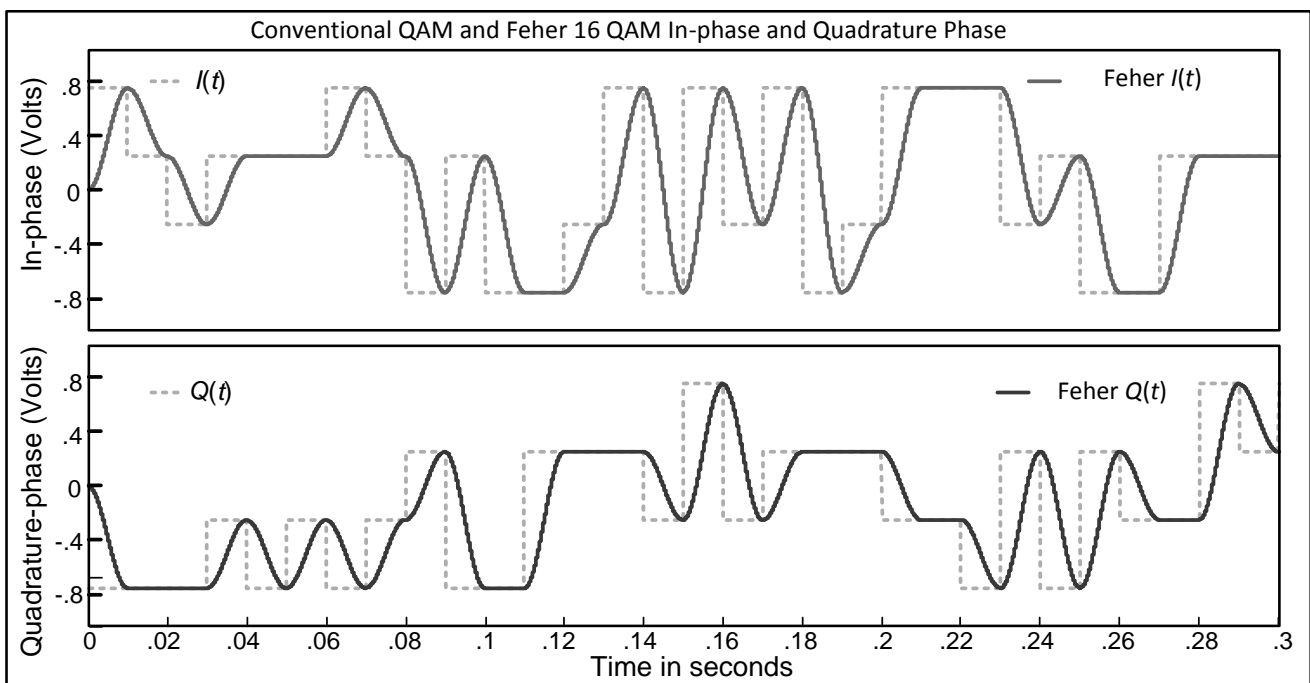


Figure 4.2. Simulated QAM and Feher QAM Waveforms

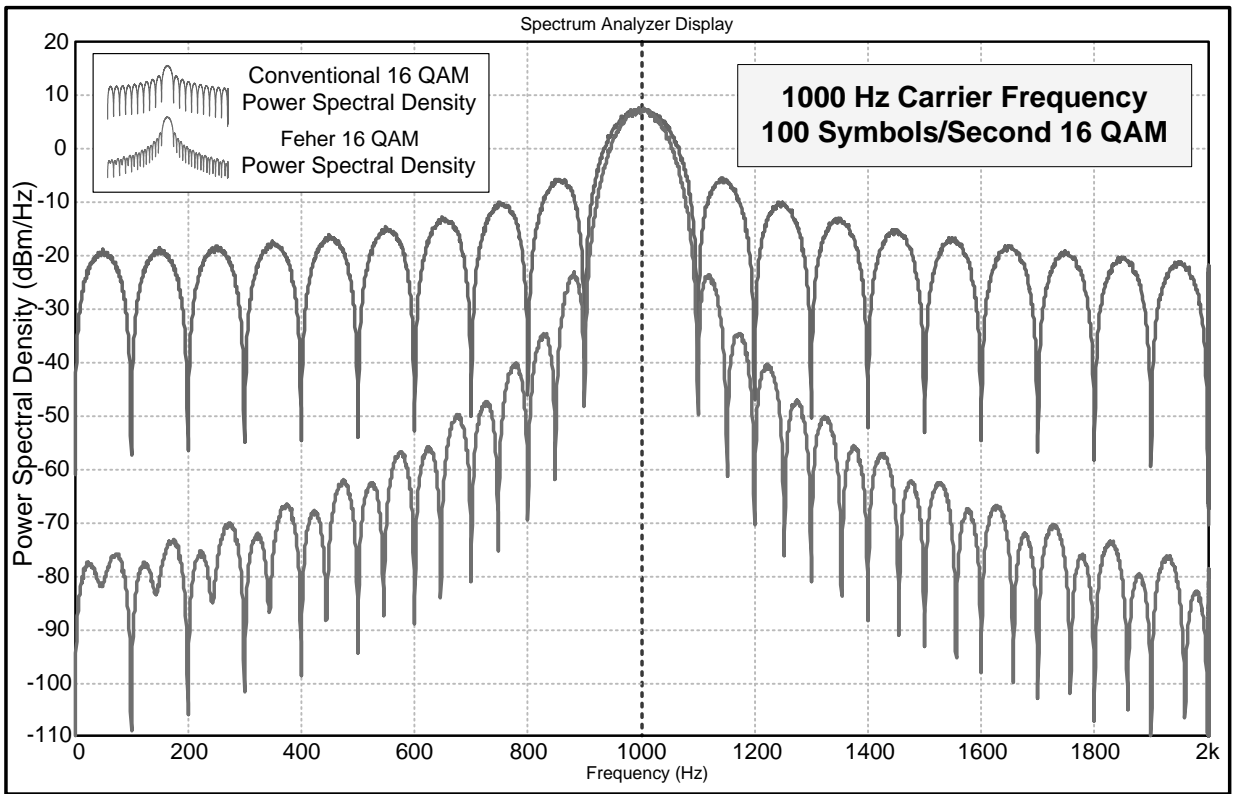


Figure 4.3. Simulated 16 QAM and Feher-QAM Power Spectral Density

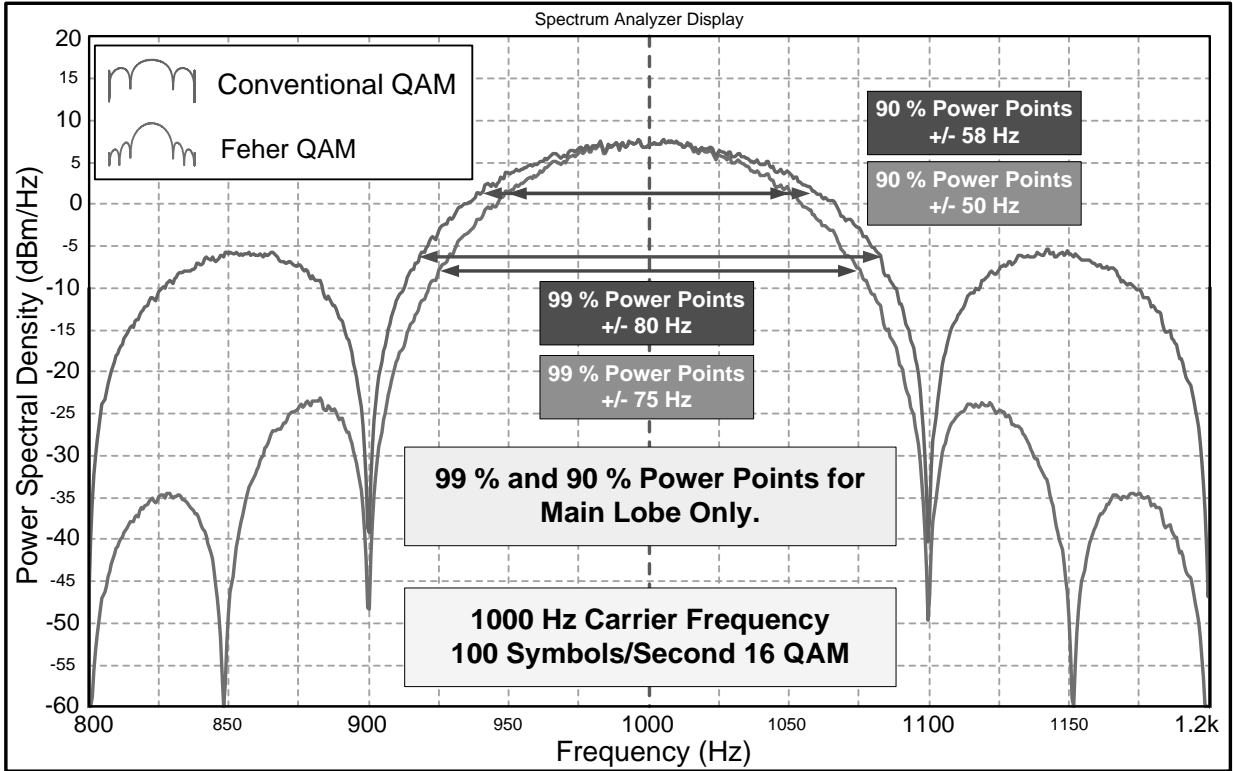


Figure 4.4. QAM and Feher-QAM 90% and 99% Power Bandwidth Points

5.0 Bandwidth Comparison

We simulated a 1000 Hz carrier frequency, 100 symbol/second conventional QAM modulator and Feher-QAM modulator using the simulation tool in [9]. Table 5.1 *only* compares power bandwidth points for the main lobes. As shown in Figure 4.4 the sidelobes for conventional QAM are much larger than Feher-QAM. Including the first sidelobes in the power bandwidth calculation would show even a *larger* improvement for Feher-QAM. Table 5.1 shows better than a 10 % improvement in the 90 % power bandwidth points for Feher-QAM. Table 5.2 shows a 30 dB improvement in power spectral density convergence at $\pm 2f_{\text{sym}}$ (2 times the symbol frequency). Tables 5.1 and 5.2, and Figures 4.3 and 4.4 show improved (reduced) bandwidth and much better convergence for Feher-QAM compared to conventional QAM.

Table 5.1. Bandwidth Points in terms of Symbol Frequency (100 Hz)

| Bandwidth Points | Conventional QAM | Feher-QAM | Improvement |
|-------------------|---------------------------|---------------------------|-------------|
| 90 % Power Points | ± 58 Hz or ± 58 % | ± 50 Hz or ± 50 % | 13.8 % |
| 99 % Power Points | ± 80 Hz or ± 80 % | ± 75 Hz or ± 75 % | 6.3 % |

Table 5.2. Convergence in terms of Symbol Frequency (f_{sym})

| Symbol Frequency | Conventional QAM | Feher -QAM | Improvement |
|-----------------------|------------------|-------------|-------------|
| $\pm 2f_{\text{sym}}$ | - 7 dBm/Hz | - 37 dBm/Hz | 30 dB |
| $\pm 4f_{\text{sym}}$ | -15 dBm/Hz | -56 dBm/Hz | 41 dB |

6. Conclusion

We show that Feher-QAM has better than a 10% improvement in bandwidth and 30 dB improvement in power spectral density at $\pm 2f_{\text{sym}}$ (2 times the symbol frequency). Feher modulation simply adds an additional DSP stage prior to the final modulation stage as shown in Figure 4.1. Feher modulation is a general technique and can easily be applied to other digital modulation techniques.

7. Acknowledgement

The author wishes to thank RDECOM community for the opportunity to research Feher modulation.

8. Disclaimer and Copyright

Reference herein to any specific commercial, private or public products, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement, recommendation, or favoring by the United States Government. The views and opinions expressed herein are strictly those of the author(s) and do not represent or reflect those of the United States Government.

US Government work. Distribution statement A: approved for public release, distribution is unlimited.

9. References

- [1] K. Feher: "Filter," US Patent 4,339,724, July 1982.
- [2] P. Simon and T. Yan: "Performance Evaluation and Interpretation of Unfiltered Feher-Patented Quadrature-Phase-Shift Keying (FQPSK)," NASA TMO Progress Report 42-137 May 15, 1999.
http://ipnpr.jpl.nasa.gov/progress_report/42-137/137C.pdf.
- [3] M. Simon and D. Divsalar: "Further Results on a Reduced-Complexity, Highly Power/Bandwidth-Efficient Coded Feher-Patented Quadrature-Phase-Shift-Keying System with Iterative Decoding," NASA IPN Progress Report 42-146, August 15, 2001. http://tmo.jpl.nasa.gov/progress_report/42-146/146I.pdf
- [4] P. Jungwirth: "SSTM," AlaSim International Conference and Exposition," Huntsville, Alabama, 6 – 7 May 2014. <http://www.almc.org/alasim-international.shtml>
- [5] P. Jungwirth: "SSTM," TAPR Conference, Austin TX, pp. 32-51, September 5-7, 2014.
<https://www.tapr.org/pdf/DCC2014-SmoothedSymbolTransitionModulation-Patrick-Jungwirth.pdf>
- [6] G. Lili, et al.: "Symmetric Raised Cosine Keying Modulation and Performance Analysis," IEEE International Conference on Computer Science and Network Technology, pp. 88-91, Dec. 2011.
- [7] M. Simon, et al.: "Trellis-Coded Quadrature-Phase-Shift Keying (QPSK) With Variable Overlapped Raised-Cosine Pulse Shaping," NASA TMO Progress Report 42-136, February 15, 1999.
ipnpr.jpl.nasa.gov/progress_report/42-136/136F.pdf
- [8] M. Austin, and M. Chang: "Quadrature Overlapped Raised Cosine Modulation," IEEE Transactions on Communications Theory, Vol. Com-29, No. 3, pp. 236-249, March 1981.
- [9] Visual Solutions: Vissim/Comm, www.vissim.com.



2015 ARRL/TAPR DCC

Update on DATV-Express exciter for Digital-ATV

by

Ken Konechy W6HHC
W6HHC@ARRL.net



DATV-Express

Abstract - The old technology of analog-ATV suffers from susceptibility to snow and multi-path ghost images. Digital-ATV (DATV) using new technologies like digital modulation, and Forward Error Correction (FEC) can result in robust video reception where analog-ATV fails, as well as providing more narrow bandwidths on the ham bands. This presentation will review progress by the DATV-Express Project Team since DCC2014. These new efforts include:

- Making the exciter more portable by Hardkernel ODROID U3 Single-Board-Computer
- Support of Narrow-BandWidth DATV down to 0.5 MHz
- Using Express_Server software to provide video by UDP
- DatvExpressServerApp software on Windows (no Linux)
- A brief report on MiniTiouner USB-based Receiver Project

DATV-Express



The Presentation Author....



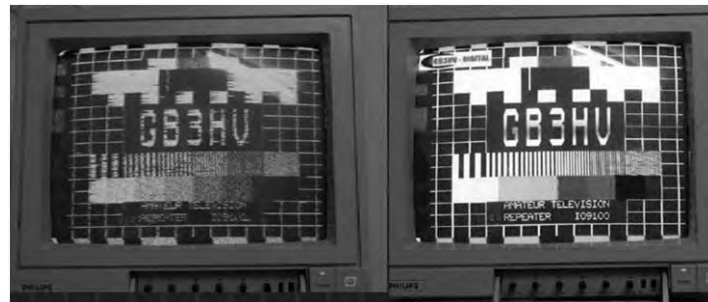
Ken W6HHC

3

DATV-Express



Digital-ATV technology allows Video Quality to exceed analog-ATV



Comparison of analog video and an DATV video using the same antennas with weak sigs

(courtesy of G7LWT & GB3HV)

4

DATV-Express



Status of Digital-ATV Today

- DATV Video Quality can exceed analog ATV
- European DATV is very active and growing
- Australia/New Zealand have lots of DATV activity
- More hams transmit DATV in USA over last 2 years
- DATV Transmitter was a cost barrier for most in USA
- Was US\$900 up for MPEG/DVB-S Encoder/XMTRs
- HiDes DATV xmitter now \$175, DATV-Express now \$300
- Lot of focus today on “ham hackable” DATV Receivers

5

DATV-Express



The DATV-Express Team

- | | | |
|-----------------|----------|-------------------|
| • Charles Brain | - G4GUO | Ferring, England |
| • Ken Konechy | - W6HHC | Orange, CA, USA |
| • Art Towslee | - WA8RMC | Columbus, OH, USA |
| • Tom Gould | - WB6P | Portland, OR, USA |

6

DATV-Express



DATV-Express Project

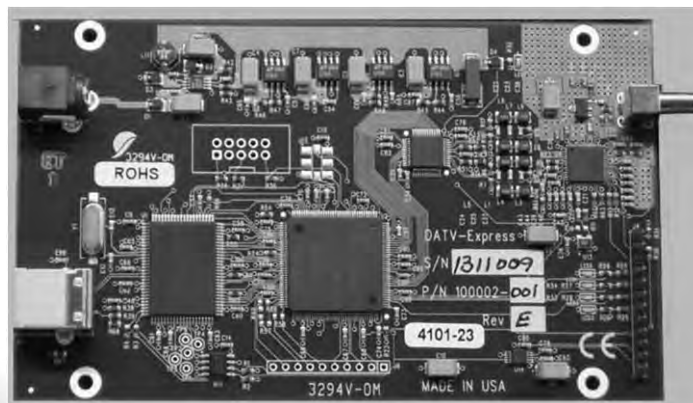
- Following 4 slides show the status at TAPR 2014

7

DATV-Express



DATV-Express SDR-based hardware board

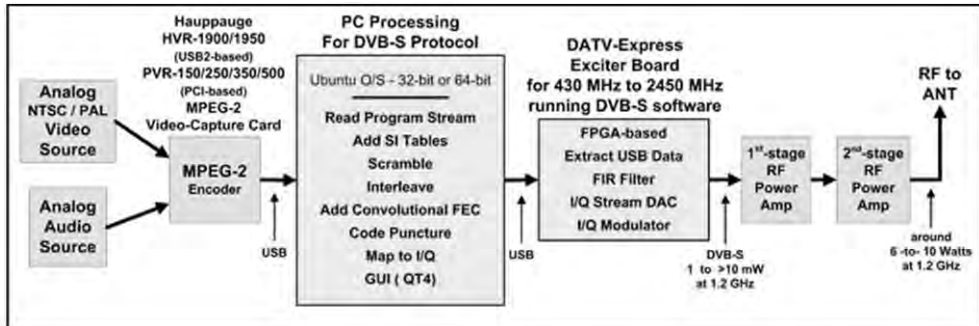


8

DATV-Express



Overview of DATV-Express System

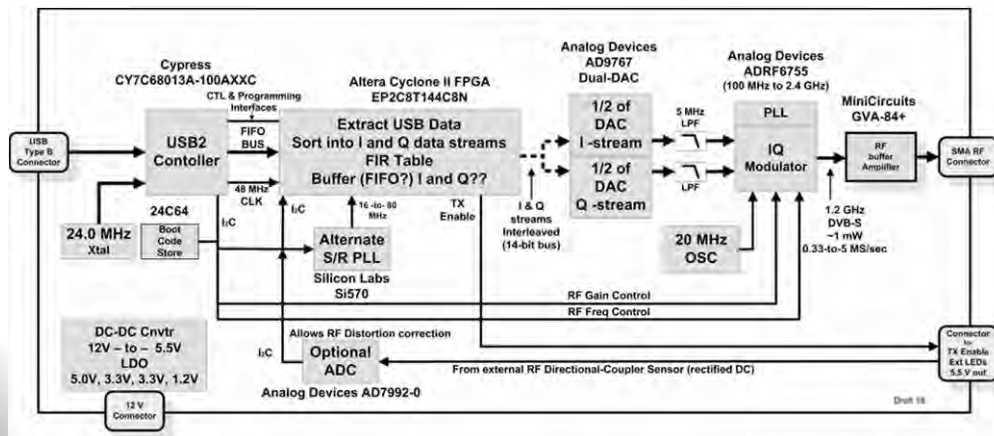


Typical System Block Diagram for DATV-Express DVB-S DATV Transmitter

DATV-Express



DATV-Express board internal block diagram



Block Diagram for DATV-Express Exciter Hardware Board

DATV-Express



DATV-Express System Specs

- DVB-S protocol is tested and released
- All IQ modulations (QPSK modulation was tested)
- Frequency Range:
70–2450 MHz (Modulator chip specification)
- Symbol-Rate:
 - Adjustable: 0.33 to 5 MSymb/second
- RF output ~ 1-20 mW buffered (SMA connector)
- USB Video Capture card for NTSC or PAL
- PC Operating System – first Ubuntu-32/64-bit

11

DATV-Express



DATV-Express Project

Five areas of progress:

- Software for quad-ARM ODROID now released
- Support of Narrow-BandWidth DATV down to 0.5 MHz
- UDP function using Express_Server software
- DatvExpressServerApp on Windows (no Linux)
- **SIDE BAR** - MiniTiouner USB-based Receiver Project

12

DATV-Express



DATV-Express software for ARM ODROID U3

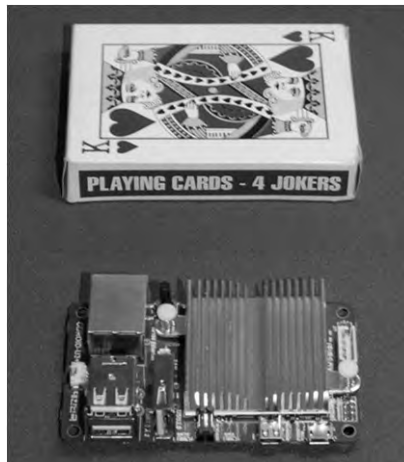
- ODROID U3 is quad-ARM “micro-PC” at 1.7 GHz
- Comes with Lubuntu 14.4 LTS (LDE Desktop)
- DVB-S protocol is now created inside FPGA (off-loads the ODROID processing load)
- ODROID prepares the Transport Stream (TS) and hands off to the FPGA
- Charles G4GUO explains that now DATV-Express project has released for ARM...it should work OK with almost any ARM product
- HardKernel has replaced model U3 with C1+ & XU4

13

DATV-Express



Hardkernel ODROID U3 “micro-PC”



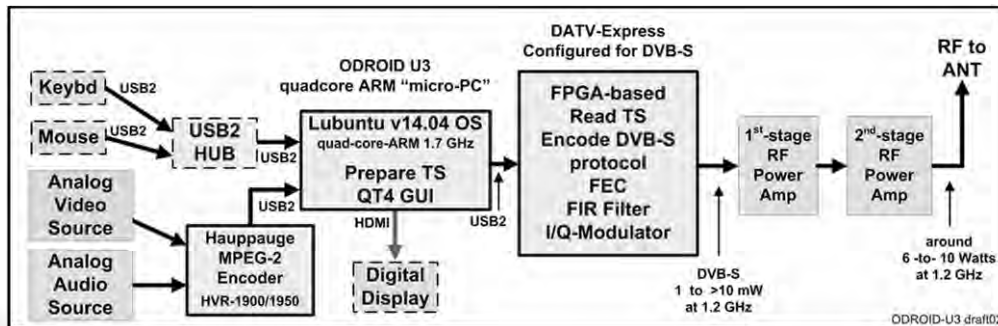
ODROID U3 is about the same size as Raspberry Pi

14

DATV-Express



Hardkernel ODROID U3



System Block Diagram for DATV-Express DVB-S with ODROID U3

15

DATV-Express



Narrow-Bandwidth DATV with DATV-Express

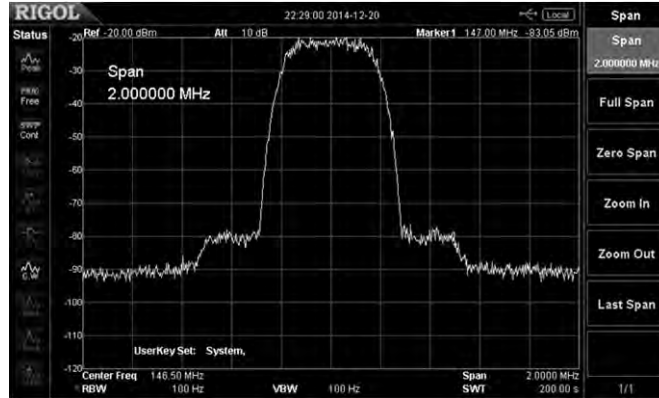
- UK OfCom has allowed temporary use DATV on 2M
- Previously unused 146.0-to-147.0 MHz now allows digital
- DATV is being sent with Symbol Rate typically 333 KSym/s
- Typically use H.264 video compression for 15 - 20 Frames/sec
- $RF\ BW_{allocated} = 0.5\ MHz$ - Typically centered 146.5 MHz
- Selectable DATV-Express FPGA code uses x64 interpolater for 100K to 400KSym/sec
- Commercial DVB-S RCVRs only go down to 1 MSym/sec
- New MiniTiouner RCVR project goes 125 KS/s to 27.5 MS/s (more details later in presentation)

16

DATV-Express



Narrow-Bandwidth DATV with DATV-Express



DATV-Express Narrow-Bandwidth DVB-S of 0.5 MHz
Spectrum Analyzer span is 2 MHz
(courtesy of G4GUO)

17

DATV-Express



UDP feature using Express_Server

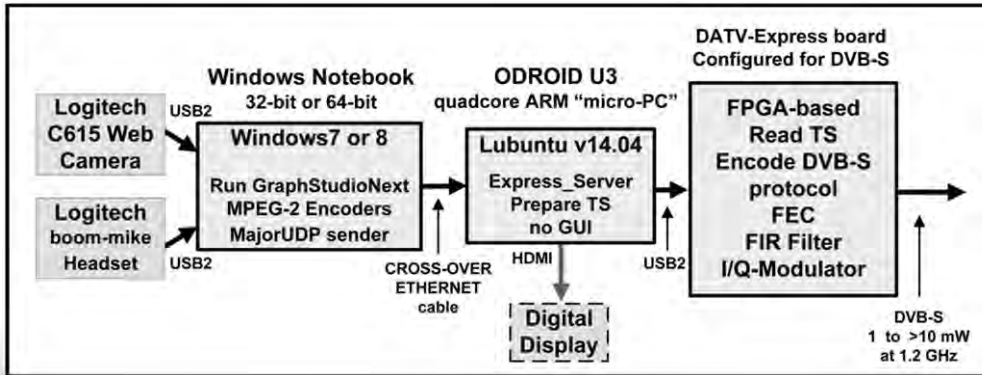
- Express_Server software was written by Charles G4GUO
- Better control for the receiving of UDP packets by the computer connected to the DATV-Express transmitter board
- Configure DirectShow filters using GraphStudioNext graphs
- Can use LogiTech C615 webcam on Windows
- MainConcepts filters provided MPEG-2 encoding
- Software encoder filters eliminate Hauppauge video-capture
- MajorUDP-Sender filter aims UDP to computer connected to DATV-Express

18

DATV-Express



UDP feature using Express_Server

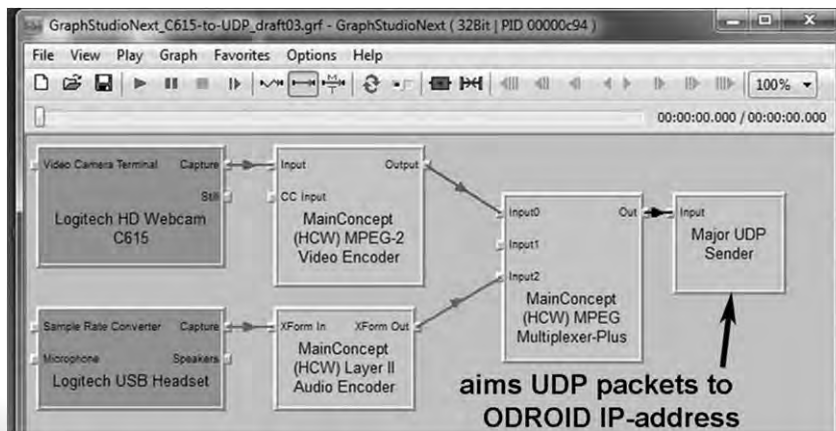


Block Diagram for sending LogiTech web cam video by UDP to ODROID running Express_Server

DATV-Express



UDP feature using Express_Server



GraphStudioNext filters for using C615 webcam on Windows MajorUDP-Sender software block is aiming packets to ODROID IP address

DATV-Express



Using DatvExpressServerApp on Windows

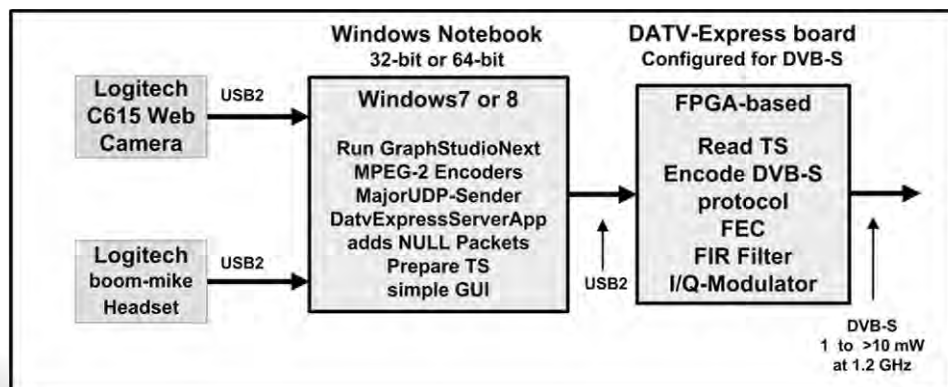
- DatvExpressServerApp software written by Charles G4GUO
- DatvExpressServerApp runs on Windows system
- **NO LINUX** involved
- Use DirectShow filters using GraphStudioNext graphs
- Can use Logitech C615 webcam on Windows
- MainConcepts filters provided MPEG-2 encoding
- MajorUDP-Sender filter aims UDP to loop-back IP-address
- DatvExpressServerApp provides a simple GUI
- DatvExpressServerApp software is still in a highly “experimental stage”

21

DATV-Express



Using DatvExpressServerApp on Windows



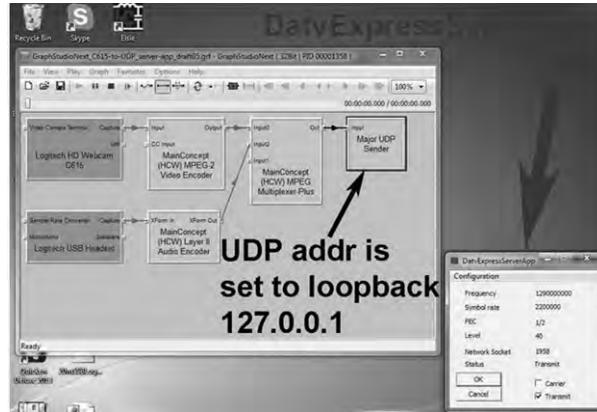
Block Diagram showing the DatvExpressServerApp software runs completely on Windows machine and connects to DATV-Express board

22

DATV-Express



Using DatvExpressServerApp on Windows



Windows running GraphStudioNext graphs and simple GUI for DatvExpressServerApp

DATV-Express



Using DatvExpressServerApp on Windows

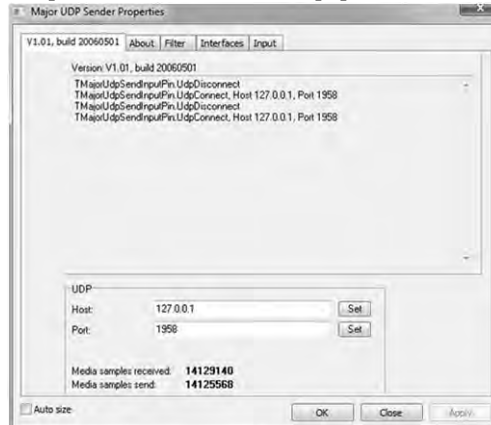


Properties of MainConcept video encoder filter using ConstantBitRate (CBR)

DATV-Express



Using DatvExpressServerApp on Windows



Properties of MajorUDP-Sender software with IP destination address aimed at loopback 127.0.0.1 and socket chosen for an arbitrary 1958

25

DATV-Express



MiniTiouner USB-based Receiver Project

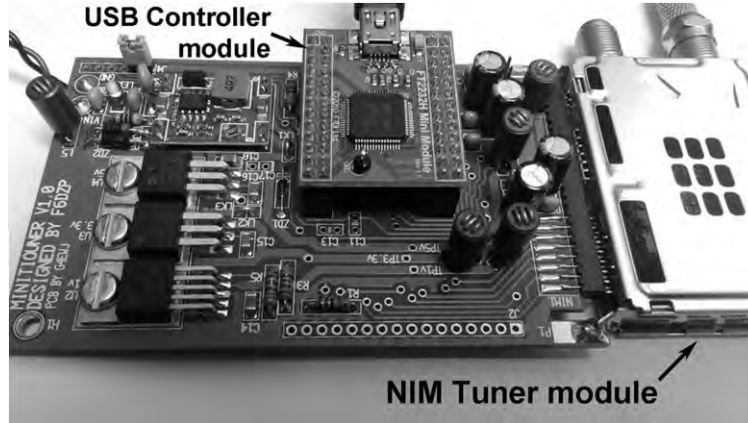
- Jean Pierre F6DZP created DVB-S/S2 analyzer software
- “Digital transmissions are not really all-or-nothing - in between there are many things that can happen” – F6DZP
- Original TuTioune software used PCI-based hardware
- New MiniTiouner receiver project is USB-based
- Software is “ham hackable” to allow fitting DATV needs
- Symbol Rates can be from 125 KSymb/s to 27.5 MSymb/s
- Jean Pierre F6DZP created software and schematic design
- Brian G4EWJ prepared PCB layout and gerber files
- BATC team sells kits on BATC Online Store

26

DATV-Express



MiniTioner USB-based Receiver Project



MiniTioner USB-based Receiver is “ham hackable”
(photo courtesy of G4KLB)

27

DATV-Express



MiniTioner USB-based Receiver Project



TiTione is DVB-S/DVB-S2 quality analyzer

28

DATV-Express



Conclusion and Plans

- DATV-Express is now released for ODROID ARM CPU's
- There were “handcuffs” that limited interest and applications:
 - Linux – steep learning curve or hams with “no interest”
 - NTSC/PAL cameras were old (becoming obsolete)
 - Hauppauge HW video encoders are difficult today (no linux)
- DatvExpressServerApp on Windows allows “escape handcuffs”
- New cameras (webcams, etc) can be selected for GraphStudioNext
- UDP opens many opportunities for remote video streams
- USB-based MiniTiouner RCVR project solves DATV problems
- Open project source code repository - - see URLs at end
- **PLANS ?** – “so many ideas, so little time”

DATV-Express



- British ATV Club - Digital Forum
www.BATC.org.UK/forum/
- CQ-DATV online (free monthly) e-magazine (ePub format)
www.CQ-DATV.mobi
- OCARC library of newsletter DATV articles
www.W6ZE.org/DATV/
- TAPR Digital Communications Conference proceedings (free downloads)
www.TAPR.org/pub_dcc.html
- Yahoo Group for Digital ATV
<http://groups.yahoo.com/group/DigitalATV/>
- DATV-Express project website
www.DATV-Express.com
- G4GUO github for DATV-Express source code
https://github.com/G4GUO/datvexpress_gui.git
- G4GUO github for express_server source code
https://github.com/G4GUO/express_server.git
- Hardkernel (Korea) for ODROID model U3 ARM-based “micro-PC”
www.hardkernel.com
- Jean Pierre F6DZP web site for TiTioune and MinTiouner
<http://vivadatv.org>

Measuring the Ionosphere at vertical incidence using Hermes,
Alex, and Munin Open HPSDR and Gnuradio.
Tom McDermott, N5EG, n5eg@tapr.org

Key words: Monostatic, Ionosphere, Doppler, Correlation, HPSDR, Gnuradio

Abstract

This paper describes a monostatic method for measuring the vertical virtual height and the vertical velocity of the F-layer of the ionosphere. The equipment is simple and relatively low power, it uses the Open HPSDR Hermes transceiver module, Munin broadband Power Amplifier (PA), and Alex RF filter module. The antennas consist of a 40m dipole and antenna tuner for transmit and an active receive loop antenna. The software real-time processing (reception, windowing, and correlation) is done using Gnuradio on a Linux PC, followed by post-processing using a Python program (multiple sweep integration and plotting).

Ionosphere

Figure 1 shows typical critical frequency for the E- and F layers versus local time of day. While the MUF depends on the angle of incidence to the ionosphere, the critical frequency is defined at vertical incidence. Generally measuring the E-layer requires using the 160m band (except occasionally near noon during the summer at lower latitudes it may be possible to use 80m). Measuring the F2-layer critical frequency can be done on the 80m band much of the day, and sometimes on the 40m band. When measuring the F-layer, the higher the frequency used the less the F-layer echo attenuation caused by transiting the E-layer twice.

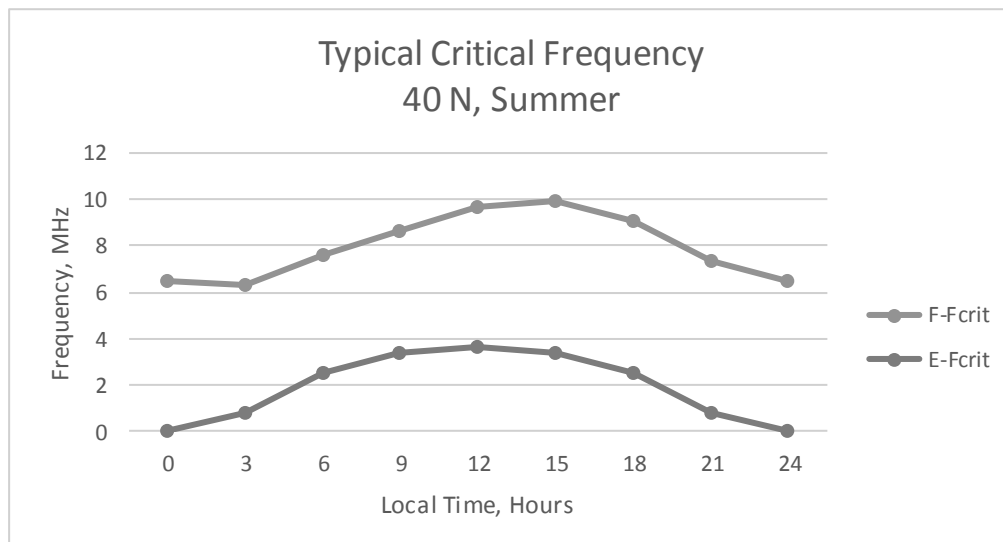


Figure 1 - Typical Critical Frequencies for E- and F- layers, summertime, 40 degrees N, Solar time, Sunspot Number = 86

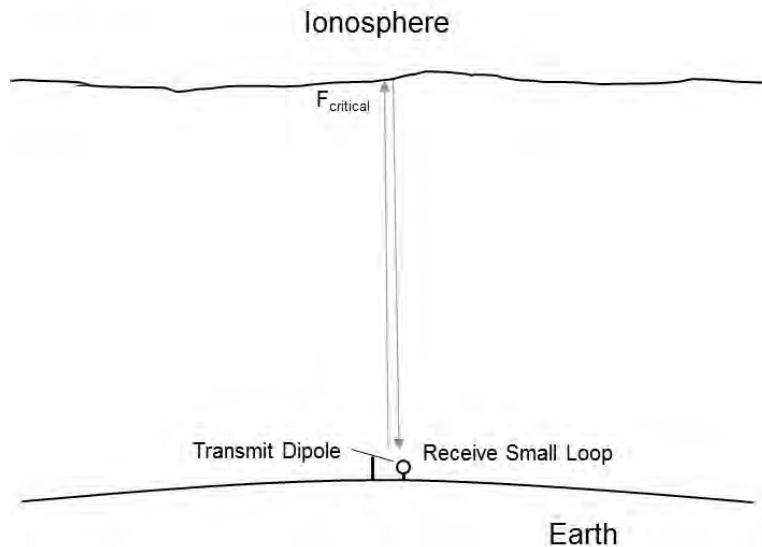


Figure 2 - Basic setup - Transmitter and Receiver Co-located.

The ionosphere reflects vertically incident signals below the critical frequency. The time-of-flight of the go plus return signal indicates how high the ionospheric layer is. Additionally the ionosphere layer may have a vertical velocity (either upwards or downwards) that induces Doppler shift onto the reflected signal. Figure 2 shows the basic setup –transmit and receive antennas are located within about 100 feet of each other – this is known as a monostatic radar configuration.

Chirp Measurement Approach

Practical measurements pose some difficult requirements because the echo is delayed less than one millisecond (E-layer) or about 1.6 milliseconds (F-layer). The monostatic approach also means that the transmit signal will be much stronger than the received signal, thus the receiver dynamic range must be large.

The approach chosen for these experiments was to transmit a linear FM chirp signal and correlate the received signal against the signal used to drive the transmitter (matched filter approach). The number of correlation taps is large in order to provide enough dynamic range and time resolution to see echos approximately 100 dB below the transmit signal. This approach requires full-duplex equipment –the transmitter and receiver are operational simultaneously. If the transmitter and receiver are co-located the transmit signal tends to overload the receiver which is listening to the same frequency at the same time as the transmitter is operating. The current experiment uses co-located Tx/Rx (same circuit board) and Tx/Rx antenna having about 20 meters separation. A similar experimental approach using 60 meters of Tx/Rx antenna separation was previously demonstrated by the Institute of Solar-Terrestrial Physics¹. The low-phase-noise and good ADC dynamic range performance of the Hermes receiver helps minimize receive noise that would otherwise obscure the desired receive echo². Chirp modulation is discussed in some recent amateur radio literature³.

A linear FM-chirp signal is a constant-amplitude signal that sweeps in frequency at a constant rate. At the end of the sweep the signal returns back to the start frequency and sweeps again. For example, an up-chirp could sweep from $-f_d$ KHz (below the channel center frequency) to $+f_d$ KHz (above the

channel center frequency), then 'snap' back to $-f_d$ kHz and start again. It's also possible to turn off the transmit signal for a short period of time during the retrace. The turn-on and turn-off parts of the signal are amplitude ramped with a raised-cosine waveform in order to prevent spectral transients. A down chirp is just the opposite, it starts at a frequency above the channel center and sweeps down at a constant sweep rate to below the channel center frequency.

The received signal is correlated against a stored version of the transmit signal. In effect the DSP correlation filter is performing as a matched filter of the chirp signal. When using a chirp to measure the ionosphere we are interested in searching for a weak replica of the chirp delayed by some amount of time related to the propagation delay, equipment delay, and frequency shifted due to the Doppler shift induced by the vertical movement of the ionosphere. Doppler shift of the received echo has the effect of appearing to alter the time delay (and thus the virtual height measurement) of the received signal. The effect of Doppler is equal and opposite for an up-chirp signal compared to a down-chirp signal. By transmitting both kinds of chirps, and analyzing them independently, we can compensate for the Doppler-induced range (height) error and additionally, measure the amount of Doppler shift induced thus allowing computation of the vertical velocity of the ionosphere. Figure 3 shows reception of an Up-chirped signal with no Doppler shift. Figure 4 shows reception of an Up-chirped signal with (+) Doppler shift, and Figure 5 shows reception of a Down-chirped signal with (+) Doppler shift.

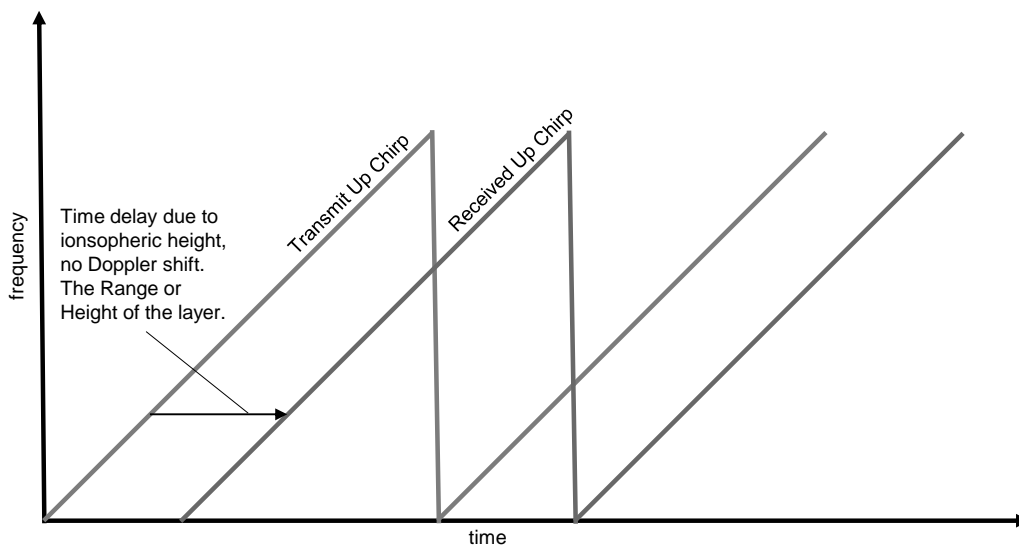


Figure 3 – Up Chirp reception with no Doppler shift.

Notice that the Up-Chirp and Down-chirp exhibit opposite range errors. This allows us to resolve the correct range and the amount of Doppler shift. The amount of range error caused by Doppler shift is dependent on the chirp rate.

The range to the ionosphere (the height) is proportional to half of the round-trip time.

$$Range (Height) = c * \frac{t_{echo}}{2} \quad (1)$$

Where c is the speed of light in m/s, and t is the time of the echo, in seconds. To compute the range error caused by Doppler, the formula is:

$$\text{Range Error} = c * \frac{d}{s} \tag{2}$$

Where d is the Doppler shift in Hz, and s is the chirp sweep rate in Hz / second.

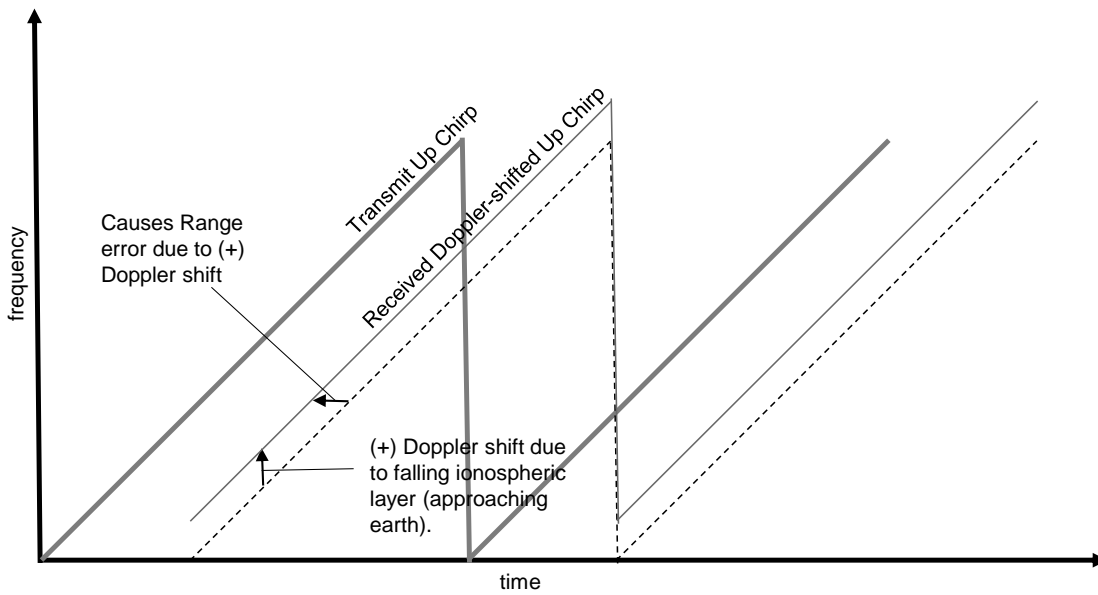


Figure 4 - Up Chirp Reception with Positive Doppler Shift due to falling ionospheric layer.

For example a chirp rate of 15,000 Hertz/second implies an equivalent range error of 19.919 km per Hertz of Doppler shift. The vertical velocity of the ionosphere is measured by the induced Doppler shift which we infer from the range error.

$$\text{Doppler Shift} = s * \frac{\text{Range Error}}{c} \tag{3}$$

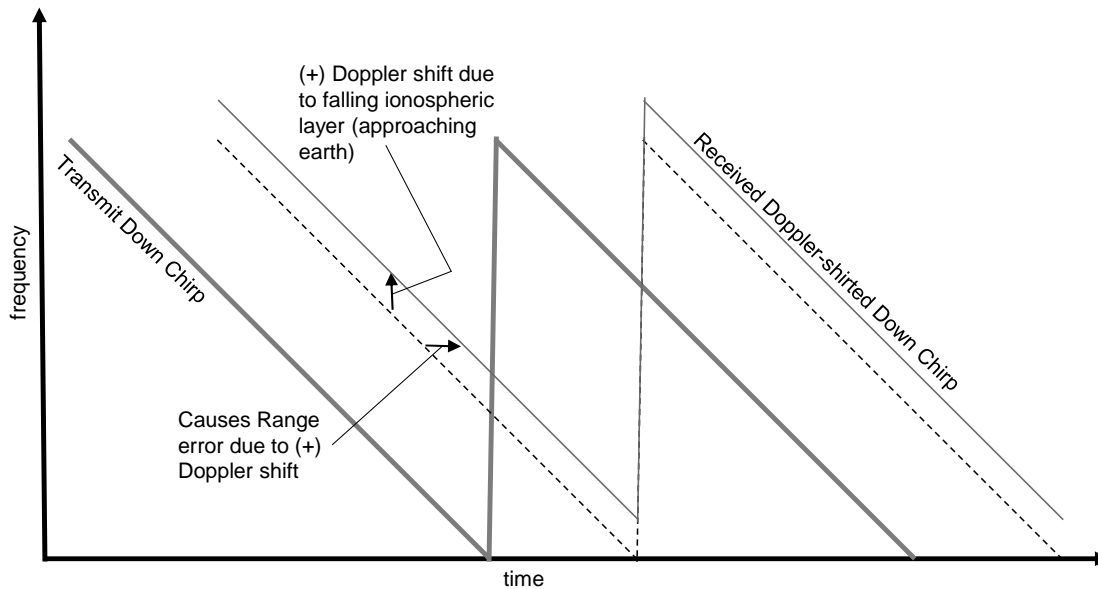


Figure 5 - Down Chirp Reception with Positive Doppler Shift due to falling ionospheric layer.

Note that the vertical movement of the layer induces a doubled Doppler shift. The chirp signal hits a moving ionosphere layer, thus being Doppler shifted as received by the ionospheric layer. Then the signal is re-emitted by the ionosphere and received back at the ground thus inducing another Doppler shift. Doppler shift is related to the layer velocity and the frequency of the radio wave, adding the factor of two for reflection from a moving ionosphere.

$$\Delta f = 2 * \frac{\Delta v}{c} * f_o \quad (4)$$

We compute the velocity of the ionosphere as:

$$\Delta v = \frac{\Delta f * c}{2 * f_o} \quad (5)$$

Relating the velocity to the measured range error:

$$\Delta v = \frac{s * Range\ Error}{2 * f_o} \quad (6)$$

We measure the range error by measuring the difference between the ranges determined by the up-chirp and the down-chirp time measurements. Since each chirp introduces an equal and opposite error, the actual range error is the difference divided by two.

$$\Delta v = \frac{s * (UpRange - DownRange)}{2 * f_o} \quad (7)$$

A slower chirp rate yields higher sensitivity to Doppler shift allowing more resolution of the ionosphere Doppler and thus the ionosphere vertical velocity.

Signal Processing

The basic receive algorithm consists of cross-correlating the received signal against a stored replica copy of the transmit signal. This is known as a matched filter. It provides a couple of benefits:

- The actual transmit signal strongly correlates with its own replica providing a convenient way to compensate for fixed equipment delays.
- A weak echo is easily seen above the background noise even in the presence of a strong transmit signal.
- The delay of the echo signal is the difference in time between the received transmit peak and the received echo peak.

This removes the requirement of knowing the absolute delay through the radio, Ethernet switch, and DSP processing – such errors or unknowns are subtracted out.

The DSP algorithm that correlates the received signal with the replica is factored through several steps in order to improve the computational efficiency. We define the two signals, $f(t)$ (transmit replica copy) and $g(t)$ (received signal). The circle-plus means convolution, and the circle-cross means correlation. The convolution of $f(t)$ with $g(t)$ is defined as:

$$f(t) \oplus g(t) = \int_{-\infty}^{+\infty} f(\tau) * g(t - \tau) d\tau \quad (8)$$

Convolution is implemented in DSP using an FIR filter kernel. A ready-made DSP block exists within Gnuradio that directly implements an FIR filter. Correlation is closely related to convolution, the filter taps need only be time-reversed to implement cross-correlation. The cross-correlation of $f(t)$ with $g(t)$ is defined as:

$$[f \otimes g](t) = \int_{-\infty}^{+\infty} f(-\tau) * g(t - \tau) d\tau \quad (9)$$

We need only time-reverse the stored replica copy of the transmit chirp signal before introducing it as the taps of the FIR filter. It's less efficient to time-reverse the receive signal because it would require buffering the received signal first. We don't need to write any code, we can simply use Gnuradio's existing FIR filter and just read in the taps from a file containing the previously stored time-reversed chirp signal.

It is not possible to actually compute an infinite number of taps, but some finite number of taps. Unfortunately an FIR filter requires on the order of N^2 operations, abbreviated $O(N^2)$. This means that if we try to implement a correlation of 1 million taps (10^6), the correlation operation requires on the order of one trillion (10^{12}) computation cycles which is infeasible. Both the signal and the taps are complex numbers (I and Q), requiring at least four floating-point multiplies and two additions per tap.

Fortunately there is a much more efficient way to implement the FIR filter in the frequency domain using the FFT (Fast Fourier Transform). This is called an FFT filter and it is also a built in block in Gnuradio, so we don't need to write it. The correlation can be implemented by taking the FFT of $f(-t)$ and the FFT of $g(t)$ and then pair-wise multiplying each element of f and g . Finally we take the inverse FFT of the result to get back to the time domain. This sounds more complicated than a direct FIR, but the FFT operation is extremely efficient such that the FFT filter requires far fewer computations, on the order of $O(N \log_2 N)$ operations. For a 1-million element correlation, this is on the order of 18 million operations (2 FFT and 1 IFFT) compared to one trillion operations using the direct FIR filter.

For convolution,

$$f(t) \oplus g(t) = IFFT [FFT(f(t)) * FFT(g(t))] \quad (10)$$

Similarly to an FIR filter used for correlation, the FFT filter can be used for correlation by time-reversing the waveform used for the filter taps.

$$f(t) \otimes g(t) = IFFT [FFT(f(-t)) * FFT(g(t))] \quad (11)$$

Properly constructed linear chirp signals have several interesting symmetry properties: an up-chirp signal is the frequency conjugate of the down-chirp signal. An up chirp signal is also the time reverse of a down chirp signal and vice versa. This means we don't even need to bother time-reversing the stored chirp signal. To correlate a receive echo, we just need to load the FFT filter taps (or FIR filter taps) with the opposite type of stored transmit chirp.

$$f(t) \otimes g(t) = IFFT [FFT(opposite chirp(t)) * FFT(g(t))] \quad (12)$$

This series of steps (leading to Equation 12) reduces the computational effort required to correlate the received signal against the stored transmit signal by an extremely large amount. At 384 k samples per second, a Core i7-3770 (3.4 GHz) processor can easily keep up with the 1-million point correlation in real time in Gnuradio. In fact, several can be run in parallel to directly compare various algorithm tradeoffs.

Figure 6 is a block diagram of the DSP steps implemented in the flowgraph. The FFT taps utilize a stored version of the chirp waveform created at 384 ksps, so no decimation is performed in the receive chain of the active flowgraph. There are very few steps required. The time domain output of the correlation filter is stored as a file on disk (File Sink) for later post processing in Python (receive integration).

Receive Lowpass Shaping Filter (Windowing in the Time domain)

In the previous figure the signal received from Hermes is first lowpass filtered in the frequency domain before being applied to the correlation function. This lowpass filter has a gentle shape defined with a Blackman-Harris window. The shaping in the frequency domain results in the time-domain samples in the correlation filter being windowed as though by a time-domain window because a linear chirp is being received (the frequencies at the extreme positive and negative ends of the chirp are the most attenuated). Without this windowing, spectral leakage would obscure the echos. Figure 7 shows the correlator output sidelobes with and without receive lowpass filtering. The time delay of the Blackman-Harris filtered and the unfiltered signals been approximately normalized with a delay element to roughly time-align the two to ease visual comparison in Figure 7.

Receive Integration

While the raw algorithm achieves about 110-120 dB of dynamic range, overloading of the receive antenna amplifier and other parts of the receiver degrades the dynamic range to about 90 dB. In order to bring the received signal up out of the noise, about 10 sweeps of the receive signal are recorded on disk. Then the signals are non-coherently averaged. This brings the F-layer echos clearly up out of the noise level.

Figure 6 is the Gnuradio flowgraph used to capture and real-time process the signal. A custom Gnuradio block was written to generate a programmable chirp signal. A number of parameters were included in that block to permit adjusting the frequency deviation, sweep rate, number of samples per sweep, and providing a raised-cosine start and stop shape. The Chirp block feeds the Hermes transmitter port. The HermesNB module was written to provide access to many features of Hermes and Alex, and has been previously described⁴.

The Hermes FPGA code also filters the transmit signal, it limits the maximum frequency response of the transmitter to ± 20 KHz of the transmit center frequency.

The reason that the echos are non-coherently integrated is that the Doppler shift induced onto the receive signal means that the relative phase of the receive echo changes each sweep of the chirp. If coherently integrated the echos would average towards zero. Non-coherently we just integrate the magnitude of each echo (neglecting phase). A 3 dB improvement in SNR should be possible through echo phase de-rotation and coherent integration.

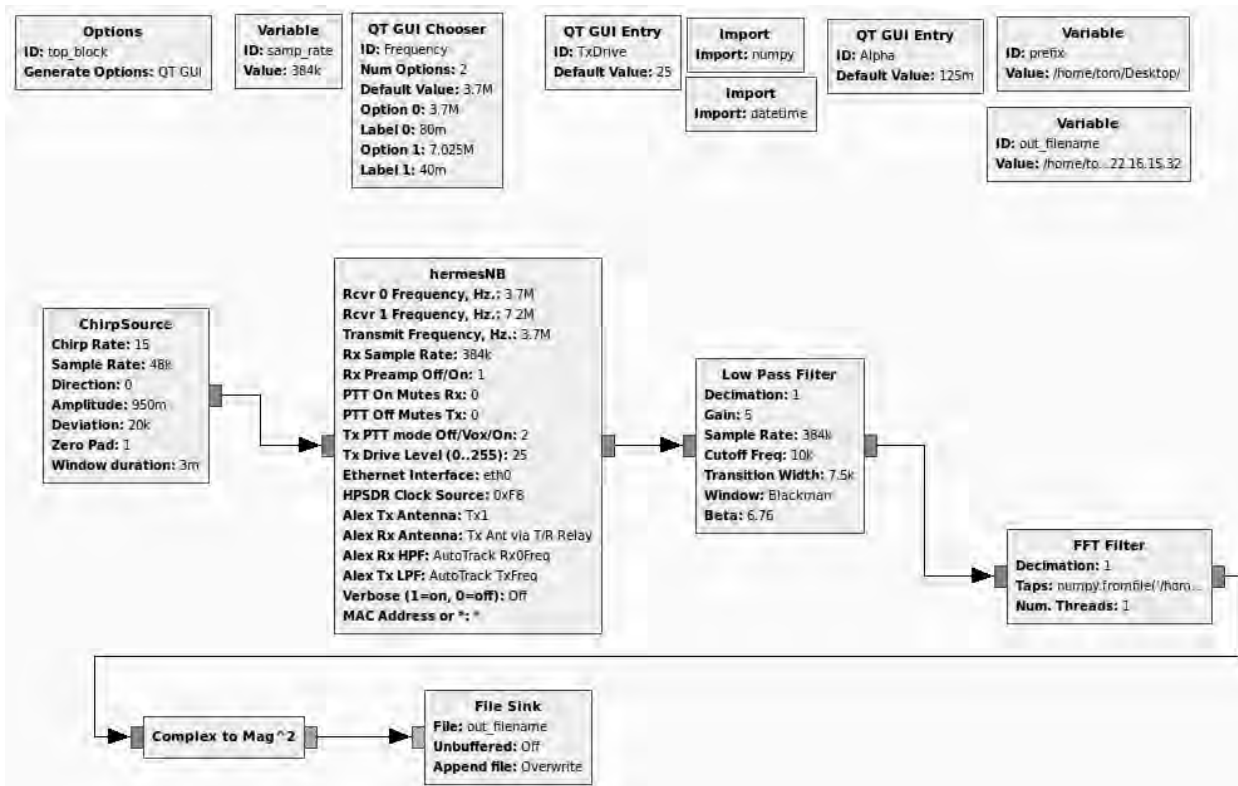


Figure 6 - Block diagram of the Gnuradio chirp transmit & receive data processing flowgraph.

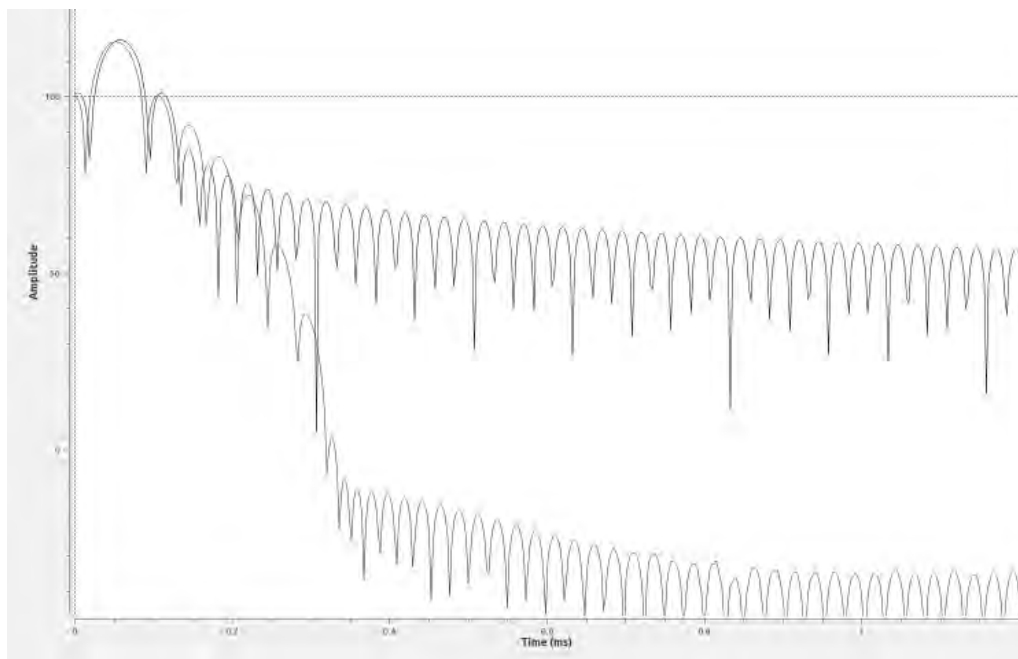


Figure 7 - Rectangular (no low pass filter) has high residual correlation sidelobes. A Blackman-Harris lowpass filter reduces the sidelobes out past about 0.3 milliseconds. The vertical scale is -50 to +120 dB.

Block Diagram

Figure 8 is a block diagram of the test setup. The Alex module is used as the transmit bandpass filter, however it is not used in the receiver path due to insufficient isolation between the transmitter and the separate receive connector on the Alex module. Input to the Hermes receiver had to completely bypass the Alex module. Gnuradio sends a FM chirp signal of constant amplitude to the transmit section of Hermes, and then to Munin (broadband PA) where it is amplified to about 20 watts output power. The amplifier output is filtered by the Alex RF filters, and sent to an antenna tuner and ladderline and a 40m dipole. For F-layer echos, the transmit signal is about 3.6 MHz.

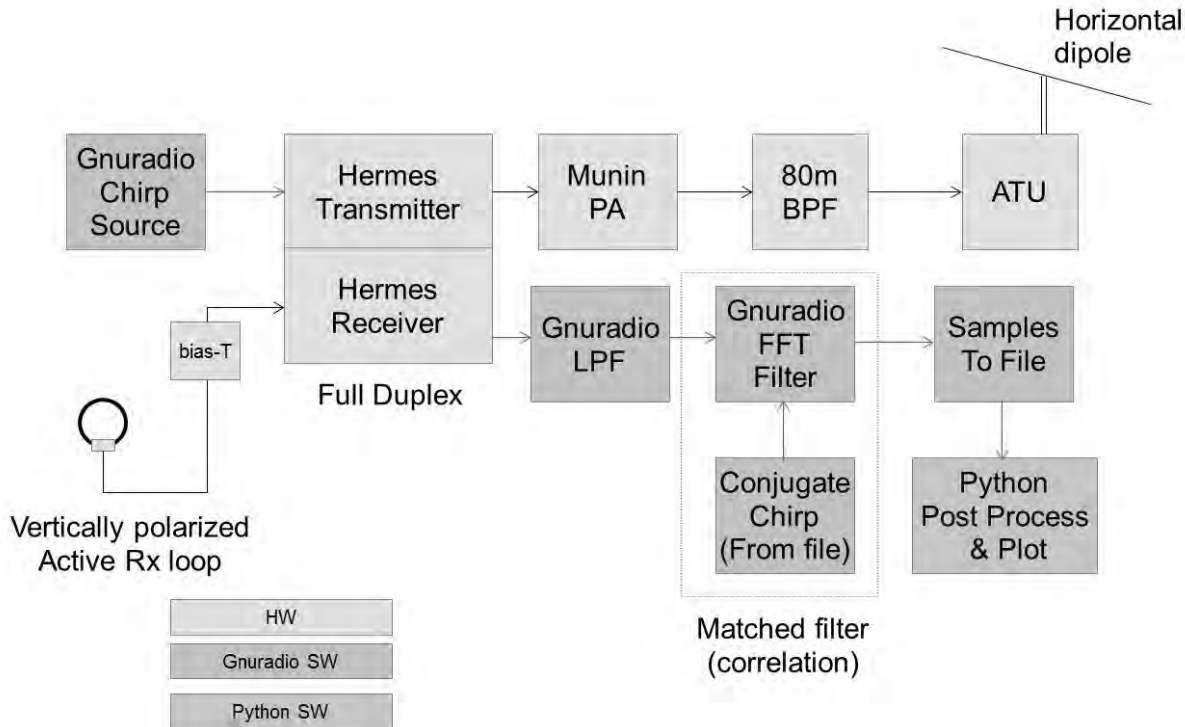


Figure 8 - Block Diagram of Test Setup. Box colors indicate functions implemented in Hardware, Gnuradio software, and Python software.

Due to the high SWR on the ladderline, about 6 watts of transmit power is actually radiated by the antenna, while 14 watts is absorbed by the feedline loss. On receive a homebrew active loop antenna 1 meter in diameter feeds a differential amplifier and balanced-to-differential transformer through a common mode choke. The antenna is remotely powered over the RG-6 feedline. After reception the Hermes signal is filtered in Gnuradio through a baseband lowpass shaping filter (to window the samples in time) before being sent to the FFT correlator.

Figure 9 is a photograph of some of the components of the experimental setup, while the RF Alex bandpass filters and the Core i7 Linux computer running Gnuradio are not shown in the photograph.

Figure 10 is a photograph of the active receive loop antenna. Only one of the two loops are used in this experiment. The loop antenna is vertically polarized, helping to reduce coupling to the horizontally polarized transmit antenna. The antenna was constructed with future experiments in mind where it

should be possible to discriminate between the Ordinary-ray (O) and eXtraordinary-ray (X) reflected by the ionosphere.

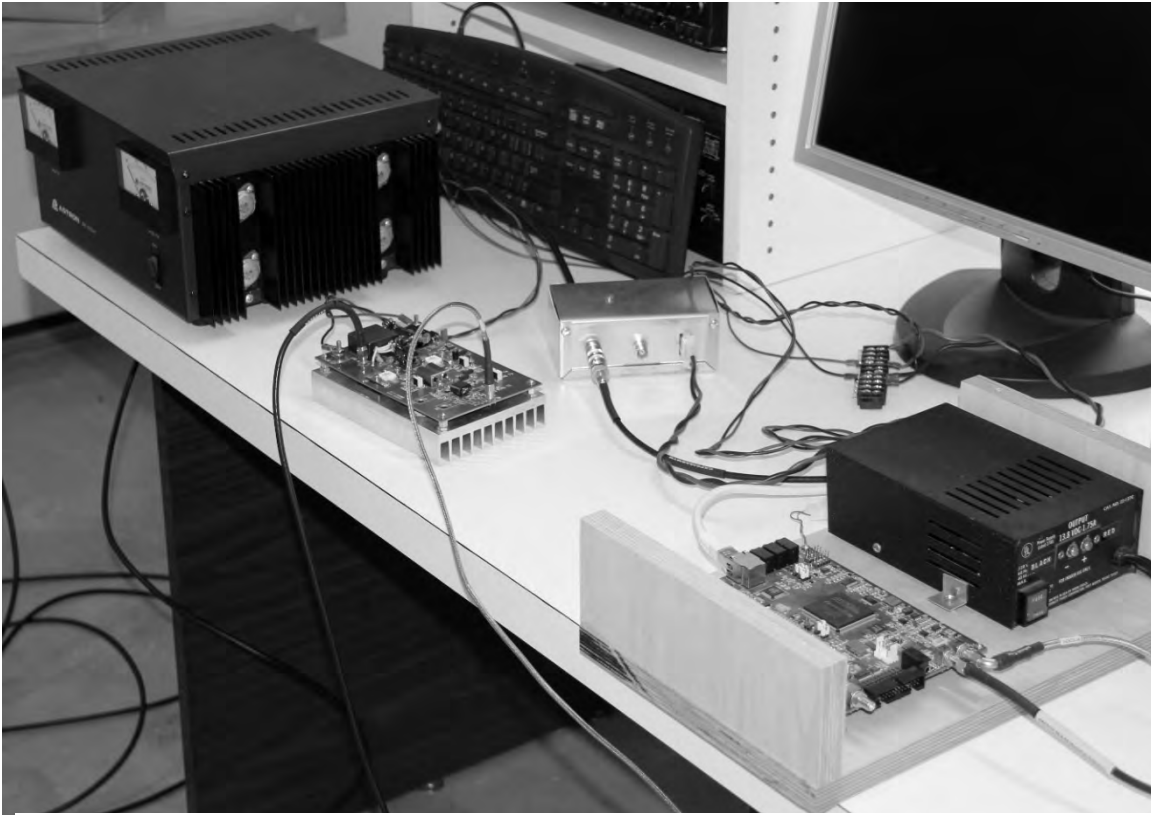


Figure 9 - Photograph of the test setup, showing DC Power Supply, Munin amplifier, Active loop bias-T, Hermes board. Not shown: Alex RF bandpass filter, Computer (outside the photo).



Figure 10 - Homebrew dual active-loop receive antenna (vertically polarized). Only one of the two loops is used in this experiment.

Results

So far measurements have been made on the F-layer at several times of day. The tests were conducted at about 3.6 MHz channel center frequency. Before dawn in fall / winter this is above the critical frequency, and no vertical reflections were received. In the evening local time the critical frequency is usually higher than 3.6 MHz, and vertical reflections have been received.

Figure 11 is a graph of the up-chirp (blue) and down-chirp (green) signals. It will be noticed that the Up- and Down- chirps exhibit range difference due to the Doppler shift of the moving ionospheric F-layer. This figure represents about ten sweeps that have been non-coherently integrated. The vertical axis is the magnitude of the correlation, in dB while the horizontal axis is time in seconds (spanning 0 to 5 milliseconds). The transmit peak has been adjusted to zero time by the post-processing Python integration program. The primary reflections are seen at about 1.7 milliseconds after the transmit peak. There is a spurious peak at 4 milliseconds (and multiples of 4 milliseconds) for both sweeps, the cause has not yet been determined.

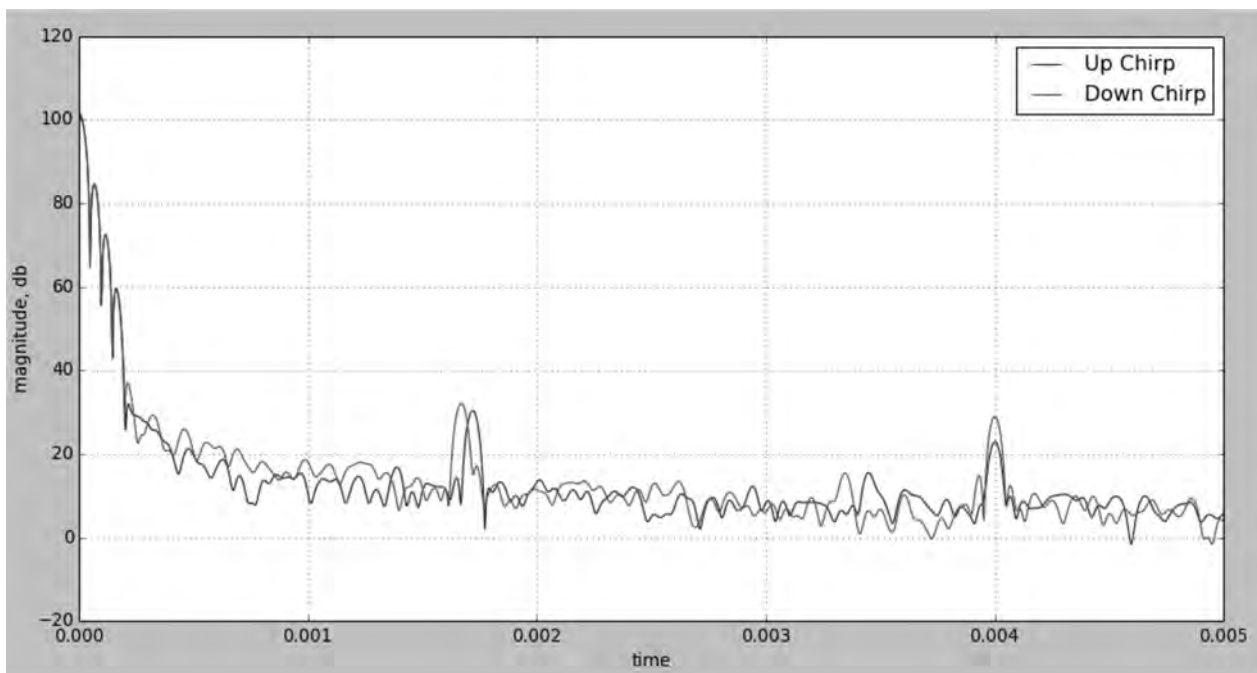


Figure 11 - F-layer reflection at 3.6 MHz in the evening local time. This figure is the correlation integral output after post-processing and integrating about 10 sweeps with Python software. The vertical axis is dB, the horizontal axis is delay time in seconds. The signals near 1.7 milliseconds are the F-layer reflections. Signals near 3.4 milliseconds are double-transit reflections. The signals at 4.0 milliseconds are system artifacts.

At about 3.4 milliseconds, small peaks in the Up- and Down-chirps can be seen. These are double-transit reflections – the signals traveled up to the ionosphere, down to ground, reflected by the ground back up to the ionosphere, and reflected back downward a second time. Notice that the Doppler range error is doubled as well.

The Doppler induced range errors indicate that the F-layer of the ionosphere is ascending at the time of this measurement.

The average of the two time measurements is the actual F-layer height, while the difference between the two is proportional to 4 times the Doppler shift. The moving ionosphere reflects the signal, thus doubling the induced Doppler shift, and the up and down chirps have opposite range errors induced. Thus the measured range difference represents quadruple the Doppler shift.

Calculations from Fig 11 show the F-layer height at 254 km, and the layer velocity at +15.4 meters per second (upward). The Doppler shift is about -0.38 Hertz. Both range and Doppler resolution is limited by the filters and the correlation width. The half-lobe width is about ± 17 microseconds, implying a range bin size of roughly 5 km.

Further Work

Figure 12 shows F-layer reflections that include reception of both the Ordinary (O) wave and the Extraordinary (X) waves. With a single receiver and linearly polarized receive antenna it is not possible to know which is O and which is X. The current single linearly polarized receive antenna sums the Right Hand Circular (RHC) and Left Hand Circular (LHC) components into one received signal. The dual-receive crossed linear loops (previously shown) plus two phase-coherent receivers (not available to the author at this time) could be used to capture two signals, one per antenna. Then the Gnuradio DSP software would be able to synthesize the right-hand and left-hand circular signals from the two linearly polarized receive signal components.

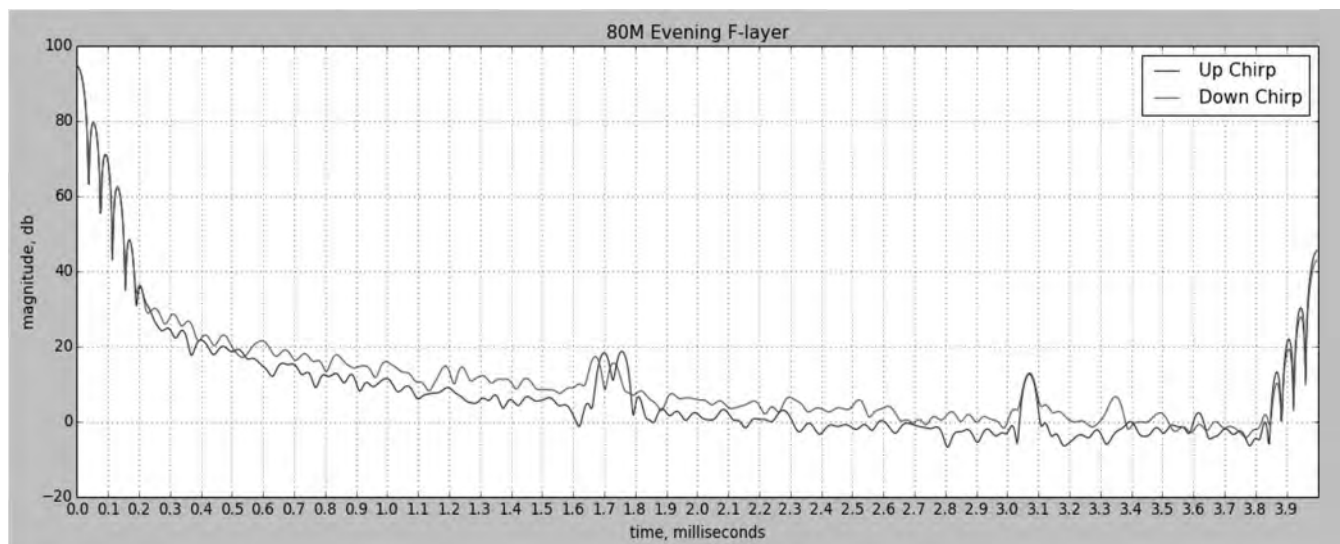


Figure 12 - F-layer reflection at 3.6 MHz showing Ordinary (O-wave) and Extraordinary (X-wave) reflection components from the F-layer near 1.7 milliseconds. The signals at 3.06 and 4 milliseconds are spurious artifacts.

Circularly polarized signals identify which reflection is O and which is X. Since the transmit antenna is linearly polarized, it emits a RHC plus a LHC signal simultaneously. These two signals remain coupled as one linearly polarized signal until encountering the ionosphere undergoing magnetic bias from the Earth's magnetic field. This causes them to de-couple into independent LHC and RHC components, which are the O and X waves. The effective ionospheric refractive index is different for the RHC and LHC components, which thus propagate with different characteristics in the ionosphere and are received as two different reflections.

We can construct the RHC and LHC components from the two received signals. Designating these two received signals as RX_a and RX_b (each complex valued) then:

$$RHC = RX_a + e^{j\pi/2}RX_b \quad (13)$$

$$LHC = RX_a + e^{-j\pi/2}RX_b \quad (14)$$

LHC and RHC can be generated at baseband using standard Gnuradio complex multiply blocks.

E-layer reflections have not been received at this time due to lack of appropriate transmit antenna.

Thanks to Andrew Martin, VK3OE, John Petrich, W7FU, and Phil Harman, VK6PH for helpful review and comments.

¹“Vertical ionosphere sounding using continuous signals with linear frequency modulation” A.V. Podlesny, V.I. Kurkin, A.V. Medvedev, K.G. Ratovsky, Institute of Solar Terrestrial Physics, Irkutsk, Russia, Conference General Assembly and Scientific Symposium 2011, Istanbul, IEEE 2011. INSPEC Accession Number: 12354018

² Phil Harman, VK6APH. 2010. “Software defined radio: The Hermes state of the art single board SDR transceiver”. RadCom 86(05):28-29.

³ Phil Harman, VK6APH. 2012. “Chirp Modulation: A sophisticated radar-like technique for propagation study that makes 100W act like 100 megawatts” RadComm 2012(3):32-38

⁴ “Gnuradio Companion module for openHPSDR Hermes / Metis SDR Radio”, Tom McDermott, N5EG, Proceedings of the 32nd ARRL and TAPR Digital Communications Conference (2013), pp 36-42.

Arduino CAT Controller for HPSDR

John Melton, G0ORX/N6LYT
4 Charlwood Close
Cophorne
West Sussex
RH10 3TG
England
john.d.melton@googlemail.com

Abstract

Simple CAT Controller for HPSDR using an Arduino micro-controller and a few switches and a step encoder.

Introduction

One of the perceived problems of the HPSDR project is that the radios do not have any knobs and buttons. PowerSDR has a CAT interface for controlling the radio that uses a serial interface. This interface is used by several add on applications that also require control of the radio, typically using a virtual serial port.

The Arduino micro-controller provides an ideal platform to build a controller with knobs and buttons. The basic Arduino UNO is cheap and provides for a number of analog and digital interfaces. Simply connecting up a few push switches and step encoders allows us to develop a custom controller.

Basic Design

A controller can be implemented using an Arduino UNO. These are readily available from several on-line resources including eBay. In the UK they can be found as cheap as £5 (\$7.50).

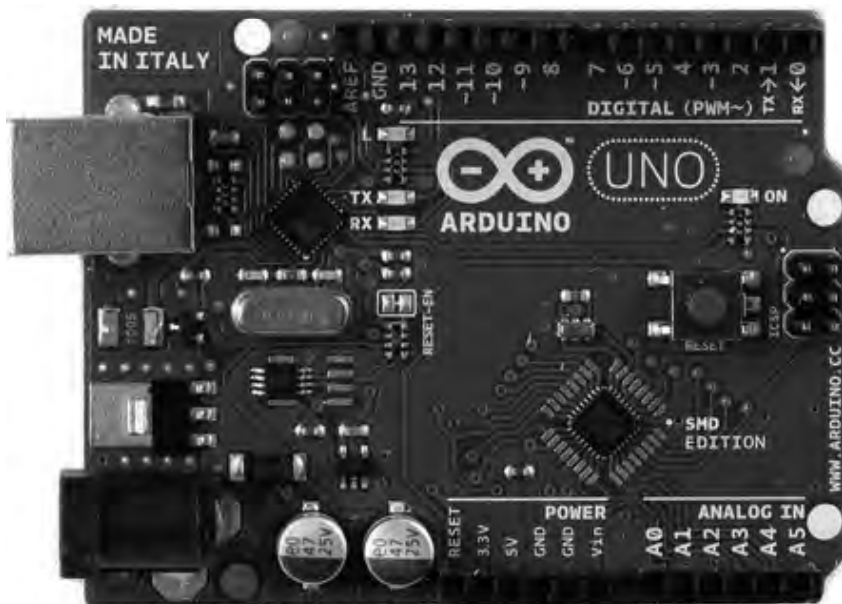


Illustration 1: Arduino Uno

The small board has a USB connector that is used for both programming the device and also as the USB Serial interface.

There are a number of digital inputs, 2 of which can be used to trigger interrupts. The digital inputs can also be programmed to enable internal pull up resistors.

There is also a power connector that is not used as the power is taken from the USB interface.

For this application we want to implement a tuning knob. We can use a step encoder. This uses 2 signal lines to send a clock pulse and a data pulse. These can be used to determine the direction of rotation. I have used a KY-040 encoder. These are available as a small board as can be seen from the image below. They also include a switch that is activated when the button is pressed. These again are readily available. I bought 5 of them for £8 (\$12) in the UK.

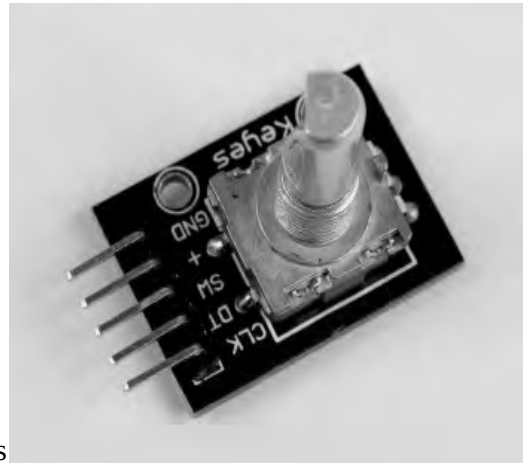


Illustration 2: KY-040 Step Encoder and Switch

There are 5 connections, CLK, DATA, SW, +5v and GND. The CLK and DATA lines had 10K pull up resistors on the board. The switch does not but if needed there are pads to add one. The encoder generates 24 pulses per revolution.

Any SPST push to make momentary switches can be used. To make connection simple I soldered a pair of header pins to each switch. For the basic controller I decided on 3 switches.



Illustration 3: Push switch with header pins

To connect up the components I used some Dupont male to female jumper cables. These usually come as a ribbon cable that can be separated.



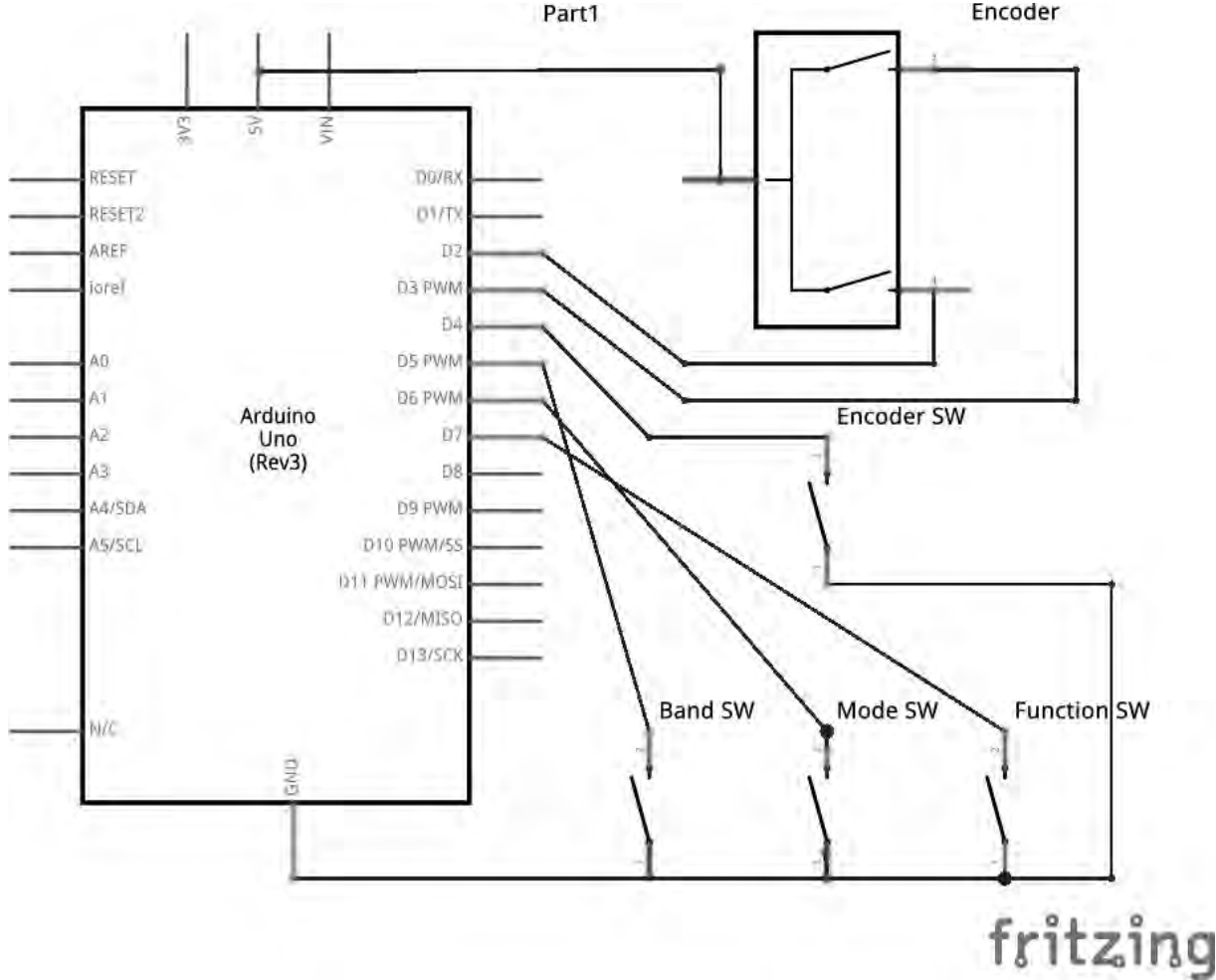
Illustration 4: Dupont jumper cables

All that is needed for the hardware is a box to mount the Arduino UNO in with the step encoder and switches mounted on the lid.



Illustration 5: Prototype Controller

Circuit Diagram



The encoders CLK and DATA lines are connected to digital pin D2 and D3 on the Arduino board. Both of these pins can trigger interrupts. This allows fast detection of the steps as it is rotated.

The switch on the encoder is connected to digital pin D4, and the remaining switches are connected to digital pins D5, D6 and D7 on one side and all to GND on the other. All the switches are configured in the software to enable the pull up resistors. This means that the by default they read a high (1) and go low (0) when pressed.

As you can see there are plenty of pins available to add additional controls.

Software

A *sketch* is the name that Arduino uses for a program. It's the unit of code that is uploaded to and run on an Arduino board. There is an IDE that is used to write the code, compile and upload to the device. A sketch always contains at least 2 functions, setup and loop.

```
void setup() {  
}  
  
void loop() {  
}
```

The code in the setup function is executed when the device is powered up or reset.

The code in the loop function is then run continually repeating the code.

A simple sketch to configure the USB serial port to run at 9600,N,8,1 and to output the state of a button connected to digital pin 3 every second is shown below:

```
const int buttonPin = 3;  
  
// setup initializes serial and the button pin  
void setup()  
{  
  Serial.begin(9600);  
  pinMode(buttonPin, INPUT);  
}  
  
// loop checks the button pin each time,  
// and will send serial if it is pressed  
void loop()  
{  
  if (digitalRead(buttonPin) == HIGH)  
    Serial.write('H');  
  else  
    Serial.write('L');  
  
  delay(1000);  
}
```

To save time and effort in development there are libraries available to handle step encoders and debounce switches. I have used Encoder and Bounce2 as they are easily available online and can be

loaded into the IDE. These are actually C++ libraries that get compiled along with the sketch.

```
#include <Encoder.h>
#include <Bounce2.h>
```

To define an encoder we simply use the constructor and pass in the pins that the clock and data are connected to. In this case they are pins 2 and 3 so the code in the library will make use of the interrupts available on those lines.

```
Encoder tuningEnc(2, 3);
```

We then need to define the switches using the debounce library. Note that at this time we do not specify the pin as this is done during setup.

```
#define encoderPin 4
Bounce encoderSwitch = Bounce();

#define bandPin 5
Bounce bandSwitch = Bounce();

#define modePin 6
Bounce modeSwitch = Bounce();

#define functionPin 7
Bounce functionSwitch = Bounce();
```

We also need to define some global variables.

```
int encoder=0; // 1 while encode switch is being pressed
int function=0; // 1 while function switch is being pressed

int afGain = -1;
int rfGain = -1;

char message[8];
int messageIndex=0;
```

The gain values are initially set to -1. This lets the code know it needs to request the current value using CAT commands.

When first powered up or reset the setup function is run. This is used to configure the digital input pins for the switches and the serial port.

```
void setup() {

  // setup function pin
  pinMode(functionPin, INPUT);
  functionSwitch.attach(functionPin);
  functionSwitch.interval(20);
  digitalWrite(functionPin, HIGH);

  // setup band pin
  pinMode(bandPin, INPUT);
```



```

bandSwitch.attach(bandPin);
bandSwitch.interval(20);
digitalWrite(bandPin, HIGH);

// setup mode pin
pinMode(modePin, INPUT);
modeSwitch.attach(modePin);
modeSwitch.interval(20);
digitalWrite(modePin, HIGH);

//setup encoder switch
pinMode(encoderPin, INPUT);
encoderSwitch.attach( encoderPin );
encoderSwitch.interval(20);
digitalWrite(encoderPin,HIGH);

Serial.begin(9600);
}

```

Once the setup function has been run the loop function is run repeatedly. We need to check the status of the switches. One switch is designated as the function switch. When held down it changes the function of the other switches. Note that as the switches are connected to ground and the internal pull up is enabled then they are indicating a high (1) value when not being pressed and a low (0) value when pressed. We process this first before checking the step encoder or other switches as its state determines their functionality. The debounce library has a call to show that the state has changed, `Bounce.update`, it returns true if it has changed.

```

void loop() {
    ...
    if(functionSwitch.update()) {
        if(functionSwitch.read()==0) {
            function=1;
        } else {
            function=0;
        }
    }
    ...
}

```

Another of the buttons is used to change the Band Up/Down. It sends the CAT command to step the band up to the next band if the function switch is not pressed and sends the CAT command to step the band down if it is pressed.

```

if(bandSwitch.update()) {
    if(bandSwitch.read()==0) {
        if(function) {
            Serial.print("ZZBD;");
        } else {
            Serial.print("ZZBU;");
        }
    }
}
}

```

As you can see, it is fairly simple to implement different CAT commands for different switches.

The tuning control simply sends a CAT command to step the tuning up/down by the current step amount as it receives each step change. The control is also used to change the audio gain if the function button is pressed or the drive level if the step button is pushed in.

One problem is that to change the audio gain or drive level the CAT command contains the value that it is to be changed to. To facilitate this, the first time the audio gain or drive level is changed, the code sees that the current value is -1 so sends a CAT command to retrieve the current value from the host software. Subsequent changes just send the new value. Note that we reset the step position to 0 each time. This means the value returned is always +1 or -1 depending on the direction.

```
long tunePosition = tuningEnc.read();
if (tunePosition != 0) {
  tuningEnc.write(0);
  if(function) {
    if(afGain==-1) {
      Serial.print("ZZAG;"); // get the current audio gain
    } else {
      if(tunePosition<0) {
        afGain--;
        if(afGain<0) {
          afGain=0;
        }
      } else {
        afGain++;
        if(afGain>100) {
          afGain=100;
        }
      }
      Serial.print("ZZAG"); // send the audio gain
      Serial.print(afGain/100);
      Serial.print((afGain%100)/10);
      Serial.print(afGain%10);
      Serial.print(";");
    }
  } else if(encoder) {
    if(rfGain==-1) {
      Serial.print("ZZPC;"); // get the current drive level
    } else {
      if(tunePosition<0) {
        rfGain--;
        if(rfGain<0) {
          rfGain=0;
        }
      } else {
        rfGain++;
        if(rfGain>100) {
          rfGain=100;
        }
      }
      Serial.print("ZZPC"); // send the drive level
      Serial.print(rfGain/100);
      Serial.print((rfGain%100)/10);
      Serial.print(rfGain%10);
      Serial.print(";");
    }
  } else {
    if(tunePosition<0) {
```

```

        Serial.print("ZZSB;"); // tuning step back
    } else {
        Serial.print("ZZSA;"); // tuning step forward
    }
}
}
}

```

The final part of the code is to handle serial data received from the host. This is implemented as a function that is called from the loop each time. This reads data 1 byte at a time and builds up the command until it sees a semicolon and then decodes the command.

```

void checkSerialData() {
    while(Serial.available() > 0) {
        // read the incoming byte:
        char c=Serial.read();
        if(c==';') {
            if(strncmp(message,"ZZAG",4)==0 && messageIndex==7) {
                int gain=((message[4]-'0')*100)+
                    ((message[5]-'0')*10)+
                    (message[6]-'0');
                afGain=gain;
            } else if(strncmp(message,"ZZPC",4)==0 && messageIndex==7) {
                int gain=((message[4]-'0')*100)+
                    ((message[5]-'0')*10)+
                    (message[6]-'0');
                rfGain=gain;
            } else if(strncmp(message,"ZZMD",4)==0 && messageIndex==6) {
                int mode=((message[4]-'0')*10)+
                    (message[5]-'0');
                if(function) {
                    mode--;
                    if(mode<0) {
                        mode=11;
                    }
                } else {
                    mode++;
                    if(mode>11) {
                        mode=0;
                    }
                }
                Serial.print("ZZMD");
                Serial.print(mode/10);
                Serial.print(mode%10);
                Serial.print(";");
            } else {
                // unhandled message;
            }
            messageIndex=0;
        } else {
            message[messageIndex++]=c;
        }
    }
}
}
}

```

Conclusion

As I have shown it is fairly simple to implement a CAT based controller for HPSDR. I have shown how to make a fairly limited, but useful, controller at minimal cost. It is not difficult to add additional buttons and step encoders to extend the functionality.

This prototype has 3 step encoders that are configured for audio gain, drive level and tuning. It also has 6 push buttons and a small OLED display. It is also using an Arduino Due for the added performance to support the OLED display.



Illustration 6: Prototype controller with more functionality

I have successfully used the controllers with openHPSDR PowerSDR, a modified version of ghsdr and a modified version of my Android client with some of the CAT commands implemented.



Illustration 7: Windows - openHPSDR PowerSDR



Illustration 8: Linux - ghsdr



Illustration 9: Android - openHPSDR

References

Arduino: <https://www.arduino.cc>

Arduino IDE download: <https://www.arduino.cc/en/Main/Software>

Controller source code: svn.tapr.org/repos_sdr_hpsdr/trunk/N6LYT/Arduino

Encoder library: http://www.pjrc.com/teensy/td_libs_Encoder.html

Switch debounce library: <http://playground.arduino.cc/Code/Bounce>

ARDOP (Amateur Radio Digital Open Protocol)

A next generation digital protocol for HF and VHF/UHF

Rick Muething KN6KB

Matthew Pitts N8OHU

John Wiseman GM8BPQ

Abstract:

The popularity of low cost PCs and tablets with substantial DSP processing power and an increasing awareness of digital signal processing in the amateur community have created an explosion of digital modes. Some of the challenges this poses are lack of portability, inconsistent “virtual TNC” interfaces and protocols optimized for single uses. ARDOP is a new protocol development which was targeted to address these challenges. The development started in 2014 and Alpha testing of the ARDOP_Win TNC (Windows version) was begun in April 2015. From the beginning the protocol was designed to cover a wide spectrum of amateur uses and be fully documented with open sourced code to encourage learning, experimentation, evolution and portability to other platforms both software and hardware. Key words: ARQ, FEC, 4FSK, 8FSK, 16FSK, 4PSK, 8PSK, WINMOR, cyclic prefix, bandwidth negotiation, automatic timing, open source and sound card protocols.

Introduction:

Today’s computing platforms (PCs, laptops, tablets and smart phones) pack more numeric processing capability than expensive dedicated DSP hardware of just 10 years ago. This with simple low cost sound cards/interfaces and modern radios with built in “sound cards” combine to make the setup and experimentation of software generated digital modes an important part of our amateur radio hobby. These modes range from simple keyboard and weak signal modes such as PSK31 and JT65 to more complex high speed message/file modes with the ability to automatically adapt to changing signal strength and propagation conditions. WINMOR [1] developed by one of the authors in 2008 has seen good acceptance as a low-cost Pactor alternative in various messaging systems like Winlink 2000 and BPQ32. Each generation of protocol and increase in low cost DSP equipment provides an opportunity to expand both the performance and flexibility of software controlled digital modes. But the development, optimization and support of a full featured digital protocol require a substantial contribution of time and skills that should be spread over many applications, operating systems and years of use. The development of ARDOP started with a short list of target objectives:

- Open Design promoting targeting to various computer, OS, and hardware platforms
- Wide range of bandwidths to optimize spectrum usage
- Automatic channel adaptability...ability to adjust to changing propagation and S:N
- Support for both connected **ARQ** (Automatic **R**etry **r**e**Q**uest) and multicast **FEC** (Forward Error Correcting) transmission modes.
- Minimize interference potential (bandwidth negotiation and effective busy channel detection)

- Flexible operating modes and radios. Compatibility with popular voice grade HF and VHF/UHF transceivers using modulation optimized for the frequency of use.
- Full binary transmission and support for multi-language UTF-8 character sets.

These expanded over a period of months to first a skeleton specification and finally a full detailed specification with detailed spread sheets showing the composition, bandwidth, robustness and speed of several modes across a 200 to 2000 Hz (audio bandwidth) spectrum [2]. In deriving the specification care was taken to provide avenues to encourage experimentation yet not impact the compatibility of compliant implementations.

Virtual TNC with Host concept

The experience with hardware TNCs and the portability of virtual (software) TNCs such as WINMOR has confirmed the benefit and flexibility of separating the “TNC” or modem function from the host (user) application. This promotes portability and allows the same TNC code or hardware implementation to be used (without change) in a variety of diverse applications such as keyboard clients, messaging systems, tracking functions, sounding systems, emergency beacons, etc. We chose this path to allow us to focus first on the protocol and TNC and not the final user host application. To the user the virtual ARDOP TNC operates similar to a hardware TNC and like a hardware TNC can display operating parameters or hidden away to avoid clutter. Figure 1 is an example of the ARDOP Win TNC showing a small but rich panel to display operating parameters, transmission progress and for the entertainment of the operator!

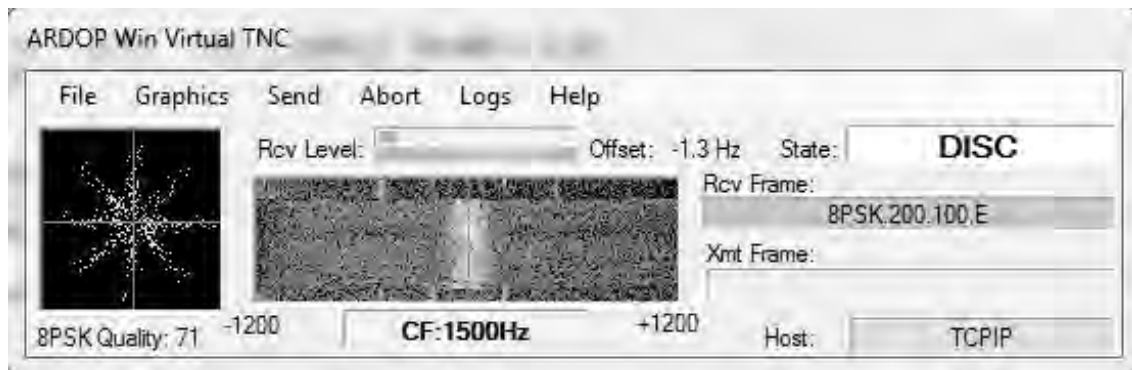


Fig 1. Screen capture of the ARDOP_Win TNC user interface/display

The virtual TNC can interface to the host program via a TCPIP connection (wired or wireless), a high speed serial connection or a wireless Bluetooth connection. This permits not only flexibility but the ability to operate the TNC/Radio combination remotely from the host software. Likewise a hardware implementation of the protocol (e.g. PIC microprocessor with sound card chip) could interface to the same host software and provide functional equivalence and compatibility. A simplified block Diagram of the ARDOP TNC is shown in Figure 2.

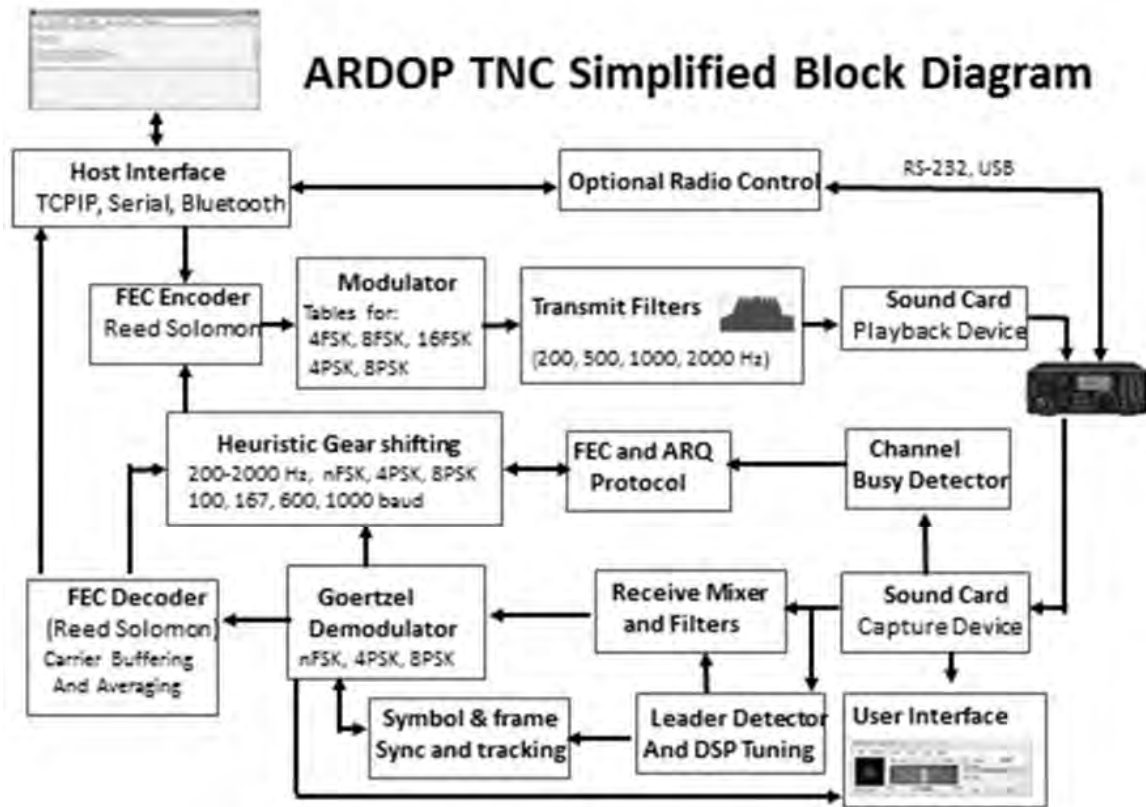


Fig 2. ARDOP TNC Simplified Block Diagram

ARDOP Performance

Most amateurs familiar with digital modes are aware of the tradeoffs required when it comes to robustness, bandwidth, signal strength and propagation quality. Some specialized modes can work deep into the negative S:N regions but they are very slow...sometimes exchanging only call signs. High speed modes permit sending large files and images but need fairly good signals, wider bandwidths and minimal multipath propagation or path compensation. Even within a 10 minute QSO or forwarding session wide variations in signal strength and propagation quality are often observed. One solution to this problem is for the sending station to adjust modulation type and bandwidth based on the *current* propagation channel. ARDOP uses a simple but effective mechanism to send the received decode quality (basically a “score” of the last received frame’s symbol constellation) back to the sending station with every ACK or NAK. During the development of ARDOP a HF channel simulator was often employed to develop the modes, mechanisms and optimum FEC level to allow the sending station to rapidly home in and maintain near optimum modulation (and bandwidth in some cases). This allows the session to proceed with the highest data rate that can be supported with the *current* S:N, propagation channel and bandwidth. Figure 3 shows two typical net throughput measurements (200 Hz channel and 2 kHz channel) made during Alpha testing using long ARQ sessions on an HF channel simulator across various HF channel types.

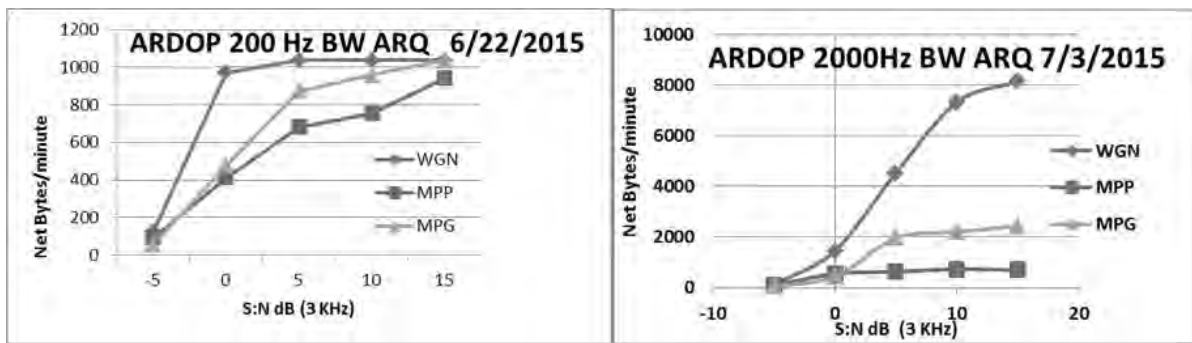


Fig 3. ARDOP performance over WGN (white Gaussian noise), MPP(multi-path poor) and MPG (multi-path good) channel types for 200 and 2000 Hz bandwidths.

Future plans include experimenting with “training sequences” and DSP path compensation techniques to allow higher performance during poor channel conditions.

Porting ARDOP to Other languages, OS and Platforms

Three significant challenges for this project from a programming perspective are as follows:

- 1) Porting the code from the initial rapid prototyping language used (Visual Basic .NET) to another language more readily usable on the various target platforms.
- 2) Targeting multiple platforms, such as Linux, Mac OS X, iOS and Android.
- 3) Finding alternatives to specialized interfaces (sound card, Internet and I/O) that were used for development of previous generations of applications by the same developers.

An interesting thing happens when you start looking into the various options and taking a hard look at the source code. For this project, conversion of the VB.NET code to C# was chosen, as a few of the source files were actually very similar to the original C/C++ code that can be found on the Internet from hams that developed software a decade or two ago. And while conversion to an alternate language may appear difficult the Internet is a good source of free tools to do rough conversions. The online code converter from Telerik [3] is the one chosen by one of the designers of ARDOP for this purpose. When the code is converted by the online tool, it is quite likely that it's not going to be ready to compile; the designer had to do a great deal of hand editing of the results to get it to compile and that is compounded by the number of files that interact in subtle ways. It also uncovers a lot of corner cases where VB.NET specific functionality has to be removed for a more workable product. This also provides an opportunity to lay the ground work for the second and third challenges; targeting multiple platforms and alternative interfaces.

When targeting multiple platforms, it is often best to understand the way each one handles user specific files such as application configuration files. It is also good to know what options exist for handling the interface to the sound hardware in the device the application will be running on. In the past, and this is often still done by application authors, configuration files have been placed in the same directory as the application executable is. This is fine in a single user environment, but not when installed on a multiuser system. The proper procedure is to place the configuration files in a folder

(also called a subdirectory) in the user's home directory. Global files with basic parameter values can be installed as well, if desired. Audio device detection can be handled one of two ways; with a custom library that is used by the software on the alternative platforms instead of the default library, or one that is available on all target platforms.

Typical Host programs

For initial on-air testing of the ARDOP protocol we needed a fairly simple host program where users could send beacons, basic keyboard text, and small files with ARDOPs FEC and ARQ modes exercising various bandwidths and modulation modes. Existing code from a prior project (V4Chat) was modified to interface to the virtual ARDOP TNC using a robust TCP IP interface. Fig 5 shows the basic ARDOP Chat host that provided setup for the ARDOP TNC, keyboard interaction, received data display and file editing and transmission along with a few conveniences like ADIF logging, beacon setup, and basic radio control (PTT and Frequency).



Fig 5. Basic ARDOP Chat host program used for initial keyboard testing.

Following initial debugging of the ARDOP Virtual TNC and ARDOP_Chat host programs John Wiseman GM8BPQ adapted his BPQ32 [4] host program to interface to the ARDOP Win TNC. This allowed additional functions including radio email and binary file transmission through the WL2K system using the existing B2 forwarding protocol. The following diagram shows how the BPQ Host interfaces to the ARDOP TNC along with conventional Packet and Pactor hardware TNCs.

This interface approach (separating the TNC DSP code from the host and interfacing through standard TCPIP, Serial or Bluetooth interfaces) allows the TNC code to be host application independent similar to the way a typical hardware TNC is.

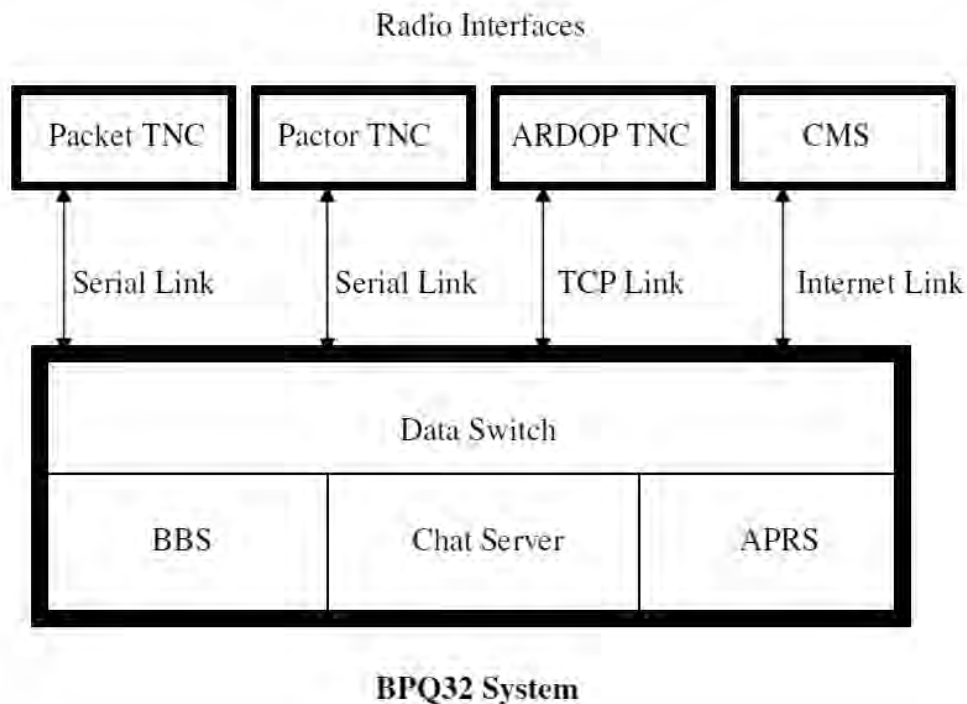


Fig 6. ARDOP Virtual TNC Interface to the BPQ32 System

Figure 7 shows a basic screen capture of an ARDOP B2F protocol session with the BPQ32 ARDOP TNC interfaced as described in Fig 6 above.

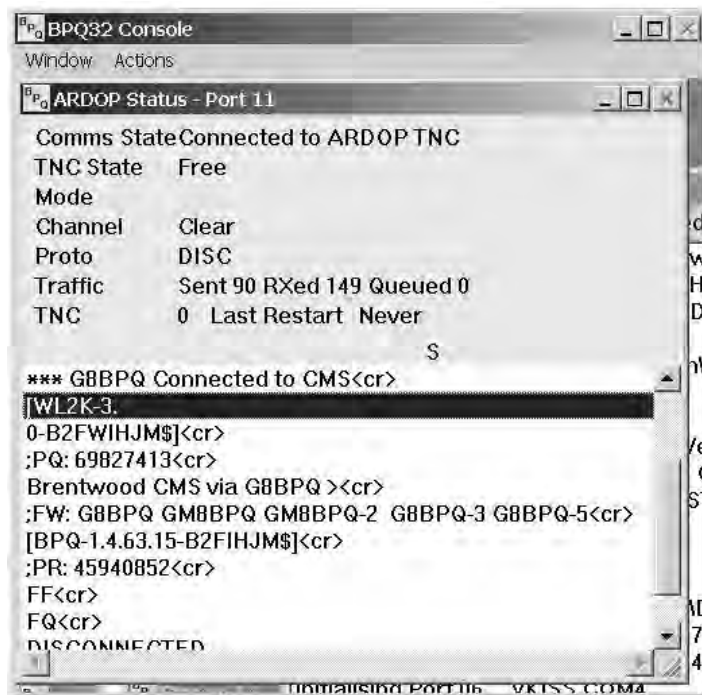


Fig 7. BPQ32 ARDOP ARQ session showing the interface to the WL2K Ham radio email system.

Project Status

The ARDOP project began Beta testing using both the ARDOP_Chat and BPQ 32 host programs in July 2015. The ARDOP Protocol spec is complete and the ARDOP virtual TNC is operational on both the Windows (Win XP- Win 10 using DirectX) and Linux (x86-Debian and ARM-Raspbian systems using MONO and the ALSA sound library) platforms and on Apple using the popular dual boot systems. A wide range of data modes covering speed and robustness ranges in excess of 40:1 are optimized for both HF (baud rate < 200 baud) and VHF/UHF FM (baud rate > 600 baud) are operational. As the Beta phase completes we will release the open source code along with a detailed testing and conformance document to allow those adapting or extending the protocol to insure basic compatibility with prior implementations. We have also initiated an effort to develop small low-cost hardware to allow wireless interfacing (Wi-Fi and Bluetooth) of small computing platforms (tablets, smart phones) to HF and VHF/UHF radios that would use the ARDOP protocol.

Credits

The authors wish to thank all those ARDOP Alpha and Beta testers from across the globe that have contributed to the development and testing of this new amateur protocol. Acknowledgement is also given to those programmers that wrote public DSP and encoding/decoding routines that were used in the ARDOP TNC. Specific reference of these is included in the commented source code.

References:

[1] (WINMOR...A Sound Card ARQ Mode for Winlink HF Digital Messaging, Rick Muething, KN6KB, 27th ARRL and TAPR Digital Communications Conference 2008)

[2] ARDOP Documentation and Code in Yahoo groups:
https://groups.yahoo.com/neo/groups/ardop_development/files
https://groups.yahoo.com/neo/groups/ardop_users/files

[3] Telerik on-line code converter. <http://converter.telerik.com>

[4] BPQ Host program by John Wiseman GM8BPQ. <http://g8bpq.org.uk>
<https://groups.yahoo.com/neo/groups/BPQ32/info>

An OS Independent and Device-Independent Mobile Web Front Panel for Radio Transceivers

Bruce Perens, Algoram
bruce@perens.com

Introduction

Algoram has produced a radio front panel that runs on almost all popular computer platforms, with only *iOS* as the exception at this time. This program is not *ported* from one system to another, the exact same code runs on every platform. It runs well on smartphones. The radio includes a WiFi access point and uses this means to communicate with the smartphone. Bluetooth can also be used.

The computing resources required in the radio to support this system are very modest and run in inexpensive microprocessors without virtual

memory support. The smartphone interface includes a waterfall bandscope and can support all manner of graphical displays and controls. The smartphone user interface doesn't require much dexterity and can be easily used by most people. Tablets of various sizes are also supported and provide additional display area and ease-of-use.

This front panel is part of *Algoram Katena*, a 50-1000 MHz software-defined HT which can be programmed to communicate using many different modulations, modes, and protocols. We've previously referred to *Katena* as "Whitebox" or "HT of the Future."

Katena is a front-panel-less HT which remains in the user's pocket or on a belt, and is controlled with a smartphone, with the smartphone providing the audio input and output as well as all front-panel functions. Currently this exists as a prototype larger than an HT, which will be made available in base and mobile form factors first, and then will be further miniaturized to become a handheld device.

The key to this technology has been the use of emerging HTML5 APIs to run our software in the device's web browser. There have been previous efforts that provided receiver interfaces, with bandscope, using some form of HTML or perhaps Java. Recently, browsers have gained standardized APIs for *two-way* audio and video communications, and thus they make the microphone and camera available to the program. They were already capable of providing all manner of 2-D displays, audio output, and controls needed to operate a radio.

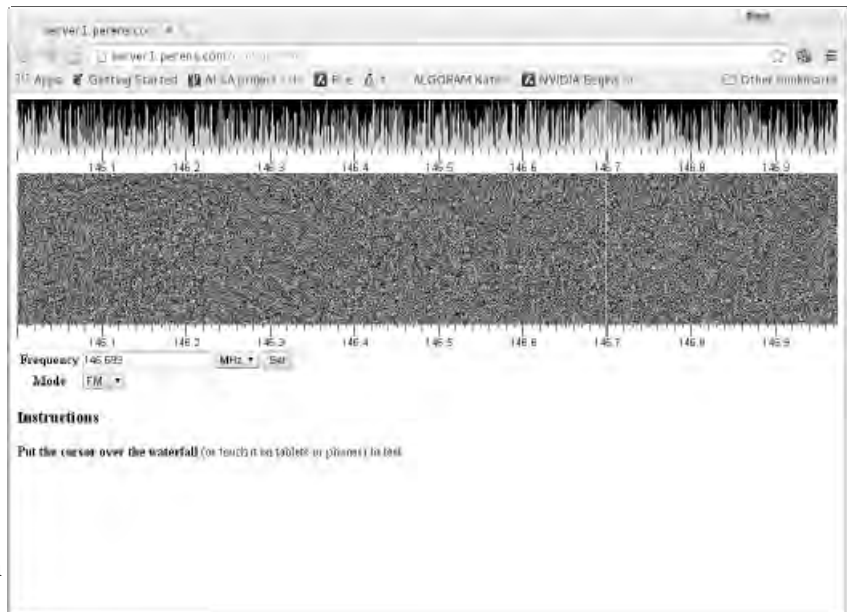


Illustration 1: A test of Algoram's waterfall bandscope using random data. Image is copyrighted by the author and released under the same terms as this paper.

Revenge of the Clones



Illustration 2: A cosplayer acting as the Star Wars clone Jango Fett.

This image was created by Sam Howzit and is under Creative Commons Attribution 2.0 license. The name and image of Jango Fett are trademarked by Disney and their use here is intended to fall under the Fair Use doctrine. Downloaded from https://commons.wikimedia.org/wiki/File:Jango_Fett.jpg

Computing hardware and operating systems are fragmentary, they aren't all the same and they don't all run the same code, and this is in general a *benefit* to society. Imagine if Operating Systems were like the “clone army” in *Star Wars* episodes II and III. Just as clone troopers would all be “identical-twin brothers” who share the same DNA, Operating Systems could all look the same and run the same code. *Wouldn't that be great?*

No. When one clone trooper got sick, they'd probably all get sick. And here's a real-world example: because *Tasmanian Devils* became so inbred that they are “clones” from an immunological perspective, they have developed a *contagious cancer* that is driving them to extinction. Cancer isn't contagious in people because we aren't clones and we each have different immune systems.

Similarly, different software doesn't fall victim to the same viruses and security bugs at the same time, and thus a network of *heterogenous* systems (ones different from each other) is more likely to have some portion continue to operate during an attack, while a network of *homogenous* systems (all the same) is likely to have all of its nodes fail.

Fifteen years ago, when the *Microsoft Windows* systems at the largest global express delivery company were attacked by the *Red Flag* virus, their entire global computer network went out of service and their hundreds of thousands of employees had to operate on phones and fax machines for a day, until the *Windows* systems could be brought down and disinfected. Only a few systems running the *BSD* operating system maintained the company's web presence.



Illustration 3: A *Tasmanian Devil* afflicted with contagious cancer. Image by Menna Jones from a PLoS paper under a Creative Commons Attribution license, see https://commons.wikimedia.org/wiki/File:Tasmanian_Devil_Facial_Tumour_Disease.png for details.



Illustration 4: “*The Scream*” by Edward Munch, has frequently been used to illustrate frustration with computer failures, although Munch did not live in the Computer Age. Copyright expired.

So, What Does This Have To Do With Ham Radio?

Hams operate *the emergency services communication network of last resort*. We are building more computers into our systems because that's the way that technology is heading, and we are creating digital networks that allow our computers to exchange data over the air, and thus make it possible for them to exchange viruses and manifest security bugs over the air. Thus, in order for our systems to be effective during emergencies, *they must not all run the same software*.

The Heartbreak of Hetrogeny

So, we've established that having *heterogenous systems*, which don't all run the same code, are important for the security and continued operations of the world's computer networks, and are even more important to the radio amateurs who operate the network of last resort. But there is also a great *cost* to heterogenous systems. Because they will not, in general, run the same code, we will often have to build separate programs to perform the same task on each different flavor of system.

Thus, a *native* application for an operating system, one which directly uses the CPU and its instruction set, the GUI, and the operating system services for that system, will be different from a native program on another OS or even a different hardware device. Native programs for Microsoft Windows and MacOS will in general look very different at the source-code level, and programs that look the same at the source code level will have to be recompiled for differing instruction sets, for example those on the the Intel CPUs common to desktops and the ARM CPUs common to smartphones.

So, we end up with a vast combinatorial problem. Even a company that can employ many programmers will find it difficult to economically support all of the available hardware and operating system combinations.

Software engineers have tried to solve this problem by making programs more *portable*, which means giving them the capability to be used on more than one kind of system. There have been many different approaches to solve this problem:

- We have Apple and Microsoft, who each would really prefer that everyone in the world run *their* systems so that there'd only be one kind of OS to program for and portability would not be an issue. But this brings us back to the “clone army” problem.
- We have *portability layers* like *wxWidgets* and *Qt*, which attempt to hide differences between systems at the source-code level, at the cost of an increase in program size and resource use, and the failure to include all native GUI and operating system facilities in its API.
- We have *Java*, which tries to hide the CPU, operating system, and GUI and run the same programs everywhere. This hasn't worked out as well as the Java designers would have liked, for example there is a different GUI on Android smartphones and desktops, even though both run Java, and because of differences in *Java* implementations “run everywhere” tends to have also meant “test every possible system”. Solving the performance issues of Java has taken the development of *just-in-time-compilers*, which turn Java into native code. These are large and consume their own resources.
- We have *software-as-a-service*, which runs the program on a server somewhere on the web, and provides it to the user via a web browser interface. This has the benefit of removing the need for users to administer servers and the programs that run on them, but it has tended to fail in a disaster, as the server is in general far away and must be accessed via a high-speed Internet connection. In a disaster, Internet access is likely to be interrupted.

Software-as-a-service can also have the effect of transmitting a service interruption far beyond the physical boundaries of a natural disaster. Hurricane Sandy took many data centers down, effecting software-as-a-service customers worldwide because not every service provider had, or could afford, fail-over mechanisms outside of the disaster area, which spanned several U.S. States.

- We have *computer languages*, many different kinds, which in general hide the differences between CPU instruction sets but not operating systems or GUIs. For example, the *C* language is available on very many different CPUs, and once you have a *C* compiler, you can use it to build the facilities of many other languages, for example *Python* and *Java*.

A New Hope

The web started out as a very simple way of displaying pages with links to other pages, but it didn't stay that way for long. The needs of providing additional interactivity, mostly to support software-as-a-service, inspired the implementation of *Javascript* (a different thing from *Java*) and the addition of many new APIs, and this continues to the present day.

On the one hand, this means that web programming is architecturally messier, and more difficult, than if the whole thing had been designed at the same time. Web programming now requires the use of at least *three* separate computing languages, *HTML*, *JavaScript*, and *CSS*, for the portion of a program that runs in a web browser, often a *fourth* language is used to implement the server-side software, and there can be even more languages involved, for example *SVG* which is used to define resolution-independent vector images, and *MathML* to format mathematical equations. There are also dozens of APIs to learn, as shown in the illustration. To handle all of these facilities, web browsers have gotten huge, and use substantial resources.

HTML5

Taxonomy & Status (October 2014)

- Recommendation/Proposec
- Candidate Recommendation
- Last Call
- Working Draft
- Non-W3C Specifications
- Deprecated or Inactive

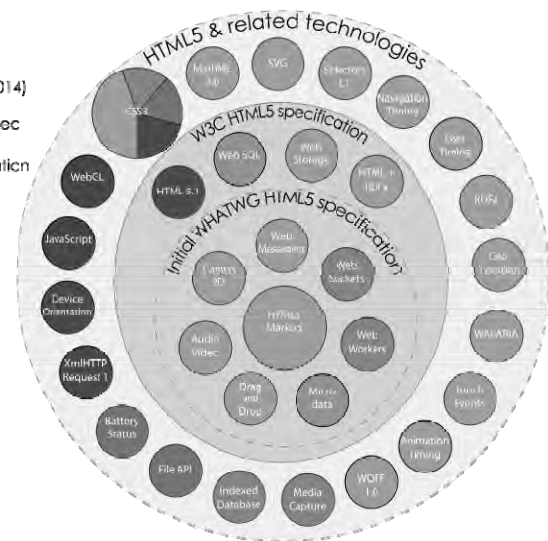


Illustration 5: HTML5 and related APIs. This image was created by Sergey Mavrody and is under the Creative Commons Attribution Share-Alike 3.0 Unported license. See https://en.wikipedia.org/wiki/HTML5#/media/File:HTML5_APIs_and_related_technologies_taxonomy_and_status.svg

On the other hand, the web browsers that closely track the HTML5 standards process, Google's *Chrome*, the Mozilla Foundation's *Firefox*, and Opera Software's eponymous browser, are now capable of all operations that we would want from a radio control panel. Most importantly, they handle two-way audio and video, 2-D graphics sufficient for radio controls and displays, and efficient network communications. These three browsers run on very many systems, and all three will run the same program. Each browser is itself built from a different code base, although some code is common to two of the three. So, there is some protection from bugs effecting all three browsers, although bugs in the user's program could be exploited on all three platforms.

The Party Pooper

Chrome, *Firefox*, and *Opera* aren't the only popular web browsers. There is *Safari*, which is available

in different versions on *Mac OS X* and *iOS*. Despite Apple's greater expense relative to other products and a corps of users who are perhaps fanatically dedicated to Apple and its products, Apple hasn't kept up and isn't capable of running all of the audio APIs necessary for a radio front panel without the creation of an *iOS*-specific application to support those facilities. To make the situation worse, Apple has a policy of handicapping competing browsers which it accepts for its App Store by insisting that they run Apple's own web rendering software rather than the browser developer's usual software. This means that Chrome on *iOS* is just as crippled by Apple's failure to keep up as Safari on *iOS*. This unfortunate policy doesn't exist for *Mac OS X*: Chrome, Firefox, and Opera are fully functional on that platform.

At this writing, it is not known at present if Apple will provide the necessary APIs on its upcoming *iOS 9*. It is expected that Apple will *eventually* provide them, but this could be years in the future.

Microsoft's *Internet Explorer* tends to have inconsistent or behind-the-times implementation of new web standards, however Chrome, Opera, and Firefox all run well on Microsoft platforms.

So, What Platforms Can We Support With HTML5 Front Panels?

At present, our HTML5 radio front panel can run on these platforms. The exact same code base runs on all of them:

- Microsoft Windows systems running *Chrome*, *Firefox*, or *Opera*.
- Mac OS X systems running *Chrome*, *Firefox*, or *Opera*.
- Android smartphones and tablets running *Chrome*, *Firefox*, or *Opera*.
- Linux systems running *Chrome*, *Firefox*, or *Opera*. This includes essentially all Linux distributions, for example *Ubuntu*, *Red Hat*, *Debian*, and *Centos*, but does not include *ucLinux*, which does not provide virtual memory. However, our server-side software runs on *ucLinux*.
- Chromebooks and ChromeOS.
- Kindle Fire HD 7, but only when you install *Chrome* using the *sideload* process rather than Amazon's app store.

These probably work too, or can be made to work, because they support the necessary browsers. But we've not tested them:

- Other Kindle tablets with non-e-paper displays and current OS software and the Fire phone, but only when you install *Chrome*, *Firefox*, or *Opera* using the *sideload* process rather than Amazon's app store.
- The BSD operating system running Chrome, Firefox, or Opera
- Firefox OS and the Firefox phone.
- Ubuntu's phone platform.

What Doesn't Work, Then?

This leaves us with *iOS* as the *only* hold-out among popular computing platforms!

And of course we could write an *app* for *iOS*, but that would be pandering to Apple's bad policies. We'll wait for them to catch up with web APIs.

Can We Support Even More?

Set-top boxes and TV dongles, and the various runners-up in the smartphone and tablet market: for example Microsoft's phone platform, Symbian, Blackberry, and WebOS might support, or might be persuaded to run, a browser with the required APIs. Android APIs are supported by some set-top boxes and TV dongles, and Android programs that are not directly available in the device's app store can often be *sideloaded* onto the device. We did sideload *Chrome* onto the Kindle Fire. This circumvented the artificial limitations of Amazon's app store, which declined to offer *Chrome* for the device in favor of Amazon's less-functional *Silk* browser.

The Operating System

Our current hardware runs *ucLinux*, a compact version of the Linux operating system that runs on devices without virtual memory. We run it on an *ARM Cortex M3* CPU within the *SmartFusion II* chip, which contains our gate-array on the same die. Our CPU is a single-core 200 MHz processor, and can yet support a significant server, WiFi, Bluetooth, Ethernet, IPV4, both USB master and slave, and FLASH storage. So, we have a capable server that will fit in your pocket with the radio.

Our software is actually a form of software-as-a-service, but with the server in your pocket! Thus, there aren't the disaster-fragility problems of the usual software-as-a-service implementations. Our server remains up on battery power and communicating with local devices via WiFi and distant devices via Amateur Radio, regardless of the state of infrastructure around it.

It is expected that later devices will eventually run the full Linux system on virtual memory hardware, rather than *ucLinux*. The selection of *Cortex M3* rather than a larger CPU is due to our use of *SmartFusion II*, which provides a FLASH-based gate-array which is capable of using battery power efficiently. The similar *Igloo II* gate-array which does not provide a CPU costs as much as the *SmartFusion II*, so we essentially get the CPU for free!

The Web Communications APIs

At first, *WebRTC* appeared to be a desirable means of communicating between a browser and a radio. It's designed for audio and video telephony as well as data communications, and includes "NAT traversal" which solves problems with calls to systems on home networks from the outside. It's connected directly to the web audio and video APIs, and automatically scales the data compression and codecs used to make the best use of the available bandwidth.

What *WebRTC* lacked was a small, Open Source, embedded library that could serve it to a browser client. Our software is Open Source, and we in general prefer to use Open Source both for economic and collaboration reasons and because we *can* fix its bugs if necessary. The only Open Source software

stack available to us used a very substantial portion of the Google Chrome browser code. That was overcomplicated and would not fit in our device.

With that determined, we switched to *Websocket*, a much simpler web API for creating a data stream between a browser and another program. On the browser side, Websocket was easy to program in Javascript, requiring only a few lines of code to handle the connection.

On the server side, we made use of *libwebsockets*, a compact Open Source embedded library in the C language. This worked, and fit well in our low-resource CPU running ucLinux. Unfortunately *libwebsockets* was not as mature as we would have liked, and has required some debugging of its internals in order for it to work correctly in our application. This work was contributed back to the project. Since our company benefits from the work of thousands of Open Source programmers, it's only fair for us to join in that work on existing programs like *libwebsockets*, as well as to contribute our own new software.

The Web GUI APIs

The web GUI is built using the HTML5 *Canvas* object, and its 2-D drawing environment. This provides a Javascript API for drawing and animating all sorts of 2-D displays, buttons, and knobs, using an imaging model descended from Adobe's *PostScript*. Canvas also supports a 3-D API which we have not made use of yet.

Input comes from the keyboard, mouse, or touchscreen, and multi-touch is used to change the bandwidth of the waterfall display using a “pinch” gesture in which two fingers are used to stretch or compress the display.

Where a keyboard is available, the space bar is used for push-to-talk. On touchscreen devices we do not use push-to-talk, but a separate *transmit* and *receive* button. This works very well on smartphones. It's awkward to hold down a screen device in the way we are accustomed to holding a push-to-talk button on an HT, especially when using a smartphone. However, there is an input for a push-to-talk switch on the radio, and we can make a PTT switch available via a USB or Bluetooth peripheral. We have programmed a library to make use of USB and Bluetooth human-interface devices, and Various USB dials and pedals have been tested.



Illustration 6: A test of web push-to-talk.

Image copyrighted by the author and released under the same terms as this paper.

Web Audio

The Web Audio API is an interesting creature, providing a graph of many different audio processing nodes that can be connected to each other, including compression, gain, frequency equalization, and even a node that runs a Fast Fourier Transform. It includes a means of acquiring the system

microphone and loudspeaker and connecting them into this graph. All of this is meant to connect directly to WebRTC, which has its own nodes that work in the audio graph. Because we use Websocket, which has no such nodes, we add to this graph two nodes which process arrays of audio samples and network data in Javascript, passing the data between the web audio API and Websocket's interface to the network. Fortunately, Javascript is well-enough optimized that this runs well even on smartphones, using substantially less than the full CPU resources.

A complication of the web audio API is that it does not allow the user to select a sampling rate, but imposes its own, and this rate differs between platforms! Thus, a simple interpolation was programmed in Javascript, and is used for both audio input and output, while the audio sample rate used for network communications is set by the program. This works sufficiently well and there is still lots of CPU left, even on a smartphone.

Another complication is that Websocket does not have access to the same means of compression that would be used with WebRTC. At present we solve this by simply sending and receiving 16-bit data at 8K samples-per-second, a bandwidth that works well on Bluetooth, WiFi, and internet connections. It would be possible, although perhaps awkward, to program entire codecs in Javascript. There have been several efforts to define and implement a subset of javascript with high efficiency, which would be appropriate for codec programming.

Putting It All Together

Using all of these APIs, we have created a complete radio front panel as two programs: a client program that runs in the browser, and is coded in Javascript, HTML, and CSS, and a server program that runs in our radio device, coded in C++. The server program stores the client program on the radio, and sends it to the browser when the browser connects to the radio.

It would be awkward if it was necessary to type in a URL to connect to the radio. Indeed, the various controls of the browser aren't really necessary for our radio front panel, and our screen would be neater if we could get rid of the URL bar and the browser menus, buttons, and tabs. Fortunately, we can! There is an evolving standard for packaging web programs as Apps, which allows them to be started by touching an icon like a conventional app, and to run in full-screen mode without any of the usual browser controls visible, only the controls in our own program. Once packaged this way, web programs become indistinguishable from apps. They can be installed from an app store, or can be sent to a smartphone by our radio for direct installation.

Security

Obviously, we must not allow unauthorized individuals to control our transmitters using web interfaces. Fortunately, we have the entire set of security facilities used for other web applications: encrypted network connections over WiFi or Bluetooth, logins and passwords, etc. But this is just the start...

Authenticating Using Logbook of the World Certificates

Inspired by a paper at the TAPR conference by Heikki Hannikainen OH7LZB, and by the work of

ARRL's Logbook of the World designers, we have implemented a means for our radio to authenticate strangers on the internet as licensed radio amateurs who are allowed to control a transmitter. Individuals who have been set up by ARRL to use Logbook of the World are each sent a file containing a x.509 public-key encryption certificate. We have instructions on how to export these certificates from LoTW and load them into a web browser. Once an Amateur has done this, the browser will cryptographically authenticate itself to our radio, communicating the amateur's identity and callsign securely.

Thus, an Amateur can make the Algoram Katena radio available as a public facility on the internet, to be used by Amateurs from all countries, and can be assured that only licensed Amateurs will be allowed to connect to the radio. Since the software gives us the Amateur's call sign, it would be possible to implement a means to automatically determine what privileges an Amateur of a particular nation and license class should be granted when operating a transmitter in another nation remotely, and to disallow operation on unauthorized frequencies and modes.

In Summary

Obviously, these are features that have never existed in a walkie-talkie, and have only been partially attempted on a few experimental base stations. So, this is going to be a whole new world for Amateur Radio. The rest of our hardware and software design is equally innovative, and will be presented in

Broadband-Hamnet™



Patrick Prescott, KC1AJT

An in-depth look at mesh networking using repurposed WiFi equipment in FCC Part 97 Amateur Radio spectrum.

Seidenberg School of
Computer Science and
Information Systems

Pace University

Pleasantville, New York

pp40879p@pace.edu

<http://www.kc1ajt.com>



This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

Table of Contents

- Introduction..... 3**
 - What is Broadband-Hamnet? 3
 - What is Amateur Radio? 3
- Capabilities of Broadband-Hamnet 4**
 - Science of 2.4 GHz 4
 - Uses for Broadband-Hamnet 4
 - Emergency Use 4
- Known Implementations..... 5**
- Hands-On With Broadband-Hamnet 5**
- Continuing Work With Broadband-Hamnet..... 6**
- The Future of Broadband-Hamnet..... 7**
- References 7**

Introduction

What is Broadband-Hamnet™?

The Broadband-Hamnet website (<http://www.broadband-hamnet.org>) describes the system as the following:

“Broadband-Hamnet™ (formerly called HSMM-Mesh™) is a high speed, self discovering, self configuring, fault tolerant, wireless computer network that can run for days from a fully charged car battery, or indefinitely with the addition of a modest solar array or other supplemental power source. The focus is on emergency communications.

In its current form it is built using the Linksys WRT54G/GL/GS wireless routers and operates on channels 1-6 of the 2.4GHz ISM band, which overlaps with the upper portion of the 13cm amateur radio band. Other platforms and bands include several types of Ubiquiti equipment in the 900MHz, 2.4GHz and 5.7GHz band. Additional features let signals come in on one band and leave on another without additional configuration. All mesh nodes on all bands exchange data so long as they are within range. We will be adding support for Ubiquiti 3.4GHz gear as well.”

What is Amateur Radio?

Amateur Radio is a “service” as defined by the Federal Communications Commission. Also called Ham Radio, the service is a hobby, public service, and a method of emergency communications. Operators must be licensed through an examination process by the FCC. The FCC has granted amateur licensees use of different parts of the radio frequency (RF) spectrum ranging from 1.8 MHz to 275 GHz and beyond.

As a hobby, operators can communicate with each other on allocated frequencies both locally and internationally. Operators often try to contact others in as many counties, states, countries, or continents as they can. Methods can include voice, Morse code, and text-based modes.

As a public service, operators provide, free of compensation, communications for countless events across the country. Examples include bikeathons, marathons, charity walks, rowing events, car races, and countless others. Many of these events are held in areas where cellular phone service may be unreliable or nonexistent. Amateur radio is able to overcome this obstacle with ease.

As a method of emergency communications, amateur radio truly shows its mite. Many disasters disrupt conventional communication methods. Phone lines can go down, cellular networks can be overwhelmed, and power utilities can go down. Even police and fire radio systems can fail. What happens when a police or fire radio tower collapses? Amateur operators are ready and willing to step in regardless of the nature and severity of the emergency.

Capabilities of the Broadband-Hamnet

Science of 2.4 GHz

Broadband-Hamnet devices are typically used in the 2.4 GHz spectrum. This is because of the overlap of IEEE 802.11g wireless networks with an amateur radio allocation in the 13cm band. There are some considerations that have to be made when using this spectrum, though. First and foremost is the propagation of a signal at these frequencies. Unlike HF (3-30 MHz) signals, signals at 2.4 GHz, at the upper part of UHF (300 MHz to 3 GHz), cannot refract off of the Earth's ionosphere and travel long distances. As a direct result, the practical range of these devices is only up to about 30 miles in open space with no obstructions. However, this cannot be accomplished with stock equipment or without a tower. Standard WiFi antennas will only present up to a 6 dBi gain. Using specialized 'yagi' or parabolic dish antennas, a gain up to 23 dBi can easily be achieved. Using this model, a point-to-point link of two radio towers 30 miles apart with no obstructions and a perfect line-of-site can be achieved. A simple leaf on a tree can interrupt a signal at this frequency, so planning and calculation is a must.

Uses for Broadband-Hamnet™

A Broadband-Hamnet system is capable of countless uses for both everyday use and emergency use. The beauty of the BBHN system is that it is 100% internet, cellular, and telephone independent. These three systems have been known to be unreliable in major disasters. Since the BBHN is based on OpenWRT software, it supports any TCP/IP device. Here are a few examples of devices and services that can be networked over BBHN:

- Computers
- Servers
- VOIP phones
- Radio Repeater Linking
- IP Cameras
- Weather Stations

The system is capable of up to 54 Mbps bandwidth, so it is capable of handling most services that are needed, even live video.

Emergency Use

In a new modern era, even our emergency response requires a constant and stable stream of information. This presents many concerns to first responders. Their work is often one-sided, get in, get the job done, get out. In an era following disasters like 9/11, Hurricane Katrina, Superstorm Sandy, and many other disasters, we have learned that a proper and accurate response requires the availability of information. Using the recent and continuing events in Nepal as an example we can discuss some of the ways BBHN could be used in a critical emergency. Now close to three weeks after the initial earthquake,

internet and phone service is still unreliable. Search and rescue groups such as the Red Cross have massive databases with information on thousands of people. How does one access this information when existing infrastructure is down? A BBHN node consumes minimal amounts of power and can be effectively powered for a week on battery or indefinitely using solar panels. Why would response teams want to depend on an already-crippled infrastructure to transport such critical information? Let's look at another use – video surveillance. In disaster zones, medical facilities and food distribution are key to the mission of the area. The BBHN system can be used to serve video surveillance, monitoring the security of these locations and the accurate operation of these facilities.

Known Implementations

Through further research on this awesome system, it was discovered that it is widely used across the US and Europe. Most of these systems are used for the services discussed above. Below are just a few of few known implementations in the US. Dozens more exist around the world, but many are not publicized.

Heart O' Texas Amateur Radio Club (HOTARC) – HOTARC maintains one of the largest known mesh networks using BBHN. The mesh covers Waco, Texas and some surrounding towns. This system has also served as a development platform for BBHN for a number of years.

Phoenix HSMM-MESH – This is a mesh in the Phoenix, Arizona area. This group is comprised of hobbyists and is not affiliated with any groups.

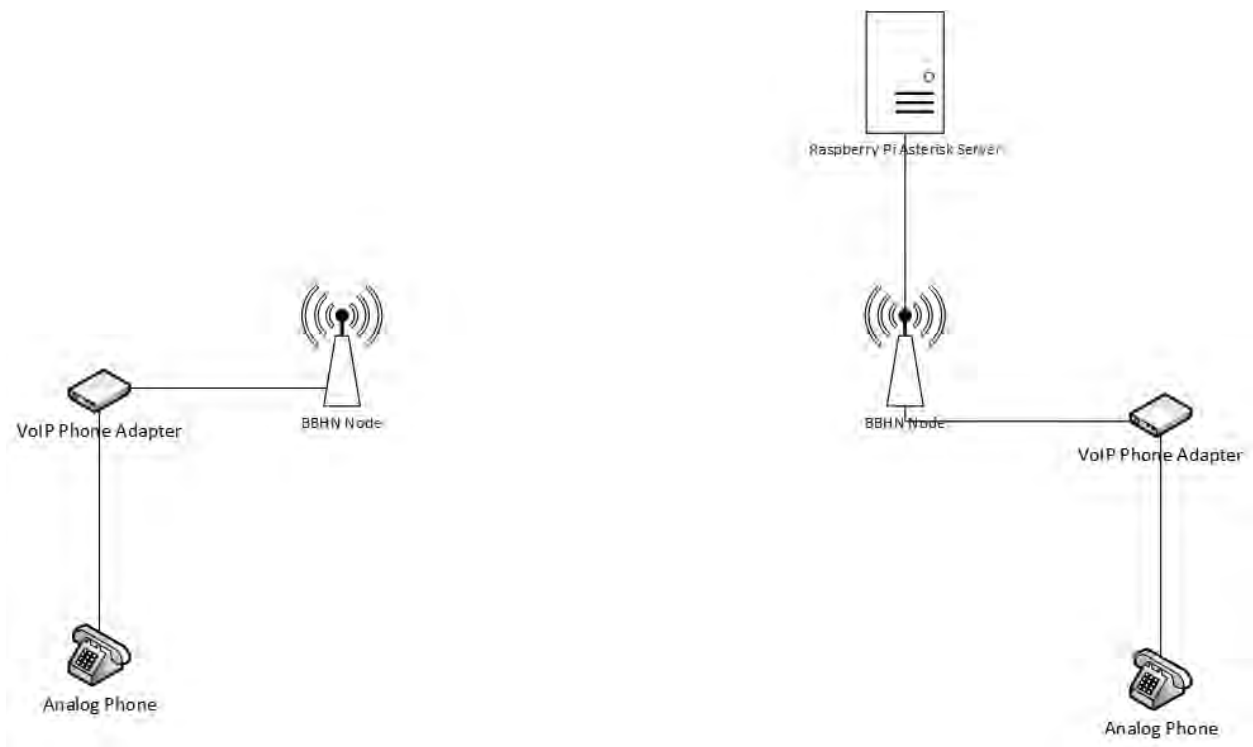
RI Mesh – This mesh is appears to also be unaffiliated. The mesh is centered around Providence and Pawtucket, Rhode Island.

Hands-On With Broadband-Hamnet

I spent a couple months working hands-on with Broadband-Hamnet and have documented my findings.

BBHN is extremely easy to use for anyone familiar with setting up wireless routers. I found that it takes about 10 minutes to get a node up and running. This includes flashing the firmware and configuring the settings as your mesh requires. The BBHN team has really great step-by-step instructions on how to get up and running. It's a very powerful system designed for use by the average person.

After getting 2 nodes configured and connected, I wanted to be able to assess their capabilities using a system that many BBHN meshes already use – Voice Over IP (VoIP). I setup a Raspberry Pi computer with an Asterisk phone server in order to get things rolling. Below is a graphic demonstrating the implementation.



I was concerned that the VoIP phone adapters would not be able to easily communicate with the server due to DNS difficulties, but there were zero problems with DNS or DHCP in BBHN. Again, this reinforces that this system is designed for the common person.

Next was voice quality, how would these calls sound? Would they sound like the other caller is in the same room or would it sound like a 1990s analog cellular phone in a tunnel? I was stunned with the quality and stability of the call. It was crystal clear. As long as the bandwidth is available and a good path can be made between the nodes, there should be no concerns with call quality.

Unfortunately, I did not have the capability to scale this test up using tower sites and greater distances. I was able to conduct a test with a 110 foot line-of-site. Again, I experienced great call quality. It's no wonder that so many mesh users like to use VoIP.

Continuing Work With Broadband-Hamnet

The conclusion of this course will not conclude my work with Broadband-Hamnet. I plan on continuing my study on the subject for the benefit of the amateur radio community. This will include a real-life implementation and a number of community publications. I strongly believe that the community can benefit from this system and improve our emergency communications capabilities.

The Future of Broadband-Hamnet

Citing current news from the BBHN website, the future of the system will be slightly altered. Recently, Linksys (a division of Cisco), announced the end of support for their WRT router series. These are the routers that BBHN was originally based on. BBHN support for these devices will not evolve beyond the current version. Future versions will only be supported by Ubiquity WiFi and mesh devices.

Updates will also only be released every 6 months. The reasoning for this comes from how BBHN works. Every node has to be on the same version of firmware in order to communicate. In order to have an emergency-ready, rapid deployment system, the need for updating should be limited as much as possible.

In the end, these devices are actually more inexpensive to implement and are designed specifically for outdoor installations. Instead of installers having to purchase cabling and an antenna, the Ubiquity devices are fully integrated and simplify the installation ten-fold. One user on the BBHN forums has already conducted a successful 32-mile point to point test with this equipment. I am confident that this is going to be a great direction for BBHN.

References

"Amateur Radio IP Networks." Amateur Radio IP Networks. N.p., n.d. Web. 15 May 2015.

<<http://www.remoteamateur.com/Home1503.aspx>>.

"Broadband-Hamnet." Broadband-Hamnet. N.p., n.d. Web. 15 May 2015. <<http://www.broadband-hamnet.org/>>.

"HOTARC Hamnet Mesh." HOTARC Hamnet Mesh. N.p., n.d. Web. 15 May 2015.

<<http://www.hotarc.org/mesh.html>>.

OpenWebRX: SDR Web Application for the Masses

András Retzler, HA7ILM

Department of Broadband Infocommunications and Electromagnetic Theory,
Budapest University of Technology and Economics, Hungary

randras@sdr.hu

Abstract—Software Defined Radio technology is getting more and more popular among amateur radio operators and hobbyists, as different universal SDR hardware devices have become available recently. OpenWebRX is a software made for those who want to set up remote SDR receivers accessible from the web. It has been developed with open-source codebase, multi-user access and easy setup in mind, to be an alternative to other similar projects (WebSDR, ShinySDR, WebRadio, etc.) It supports cheap RTL2832U based tuners. Basically OpenWebRX is an on-line communications receiver for analog transmissions (AM/FM/SSB/CW), with a web UI on which real-time waterfall display is available. Users can select a channel within the bandwidth of the sampled signal acquired from the SDR hardware. The selected channel is demodulated on the server and the resulting audio is streamed to the browser of the user, where it is played back. Users can set receiver parameters (channel frequency, modulation mode, filter bandwidth) independently. The digital signal processing functions have been placed in a separate library, *libcsdr*, which can also be considered useful as a standalone package. It performs the digital downconversion, filtering and demodulation tasks on the I/Q data.

Keywords—SDR, RTL-SDR, open-source, HTML5

I. INTRODUCTION

The speed of digital computing is increasing continuously. In addition, in the last years reasonably fast A/D and D/A converters have become available. It has become feasible to implement most of the signal processing digitally in radio systems.

Nowadays Software Defined Radio is widely used in commercial systems from satellite communication to mobile telephony, but it mostly does its task hidden in the background, replacing the conventional analog communications equipment. These embedded SDRs are tailored for a specific need (for example running a 4G base station), producing better results than their analog competitors and solving advanced problems at a lower price.

However, there is also a need for universal SDR hardware for prototyping, measurements and for use in real-world projects. While devices like the USRP, HackRF, BladeRF have come to the market to support professional and academic users, a lot of amateurs have also started to experiment with SDR technology, building cheap receivers for specific bands. When the RTL-SDR project [1] has been released in 2012, even more people got in connection with SDR directly. A mass-produced DVB-T receiver USB dongle with an RTL2832U chip is a great entry-level device that can be acquired for \$15, and with the help of the RTL-SDR project, it is possible to use it as a general purpose SDR. It tunes between 24 MHz and 1700 MHz, can be used as a receiver for a variety of wireless equipment, from two-way radios to wireless temperature sensors, and it works on multiple amateur radio bands as well. There is a vast number of free software available for decoding different signals with it. Although its professional competitors provide better performance, it is still an ideal choice for the experimenters.

RTL-SDR is also a great educational tool that can help to get more young people into amateur radio. For those, who already have the required background in IT, playing with the cheap DVB-T stick is a great chance to dive into the radio waves. I even had a friend under the age of 15 who managed to receive the signals of an amateur radio satellite with RTL-SDR on his own.

Two years ago I released an open-source project, SDRLab [2], which implements an RTL-SDR interface for LabVIEW along with a simple WFM demodulator example, and since then, I received e-mails from a few university students from different countries, asking about it. It turned out that at some universities, RTL-SDR is used as an educational tool to teach digital signal processing.

So why not bring SDR closer to even more people? The motivation behind OpenWebRX was to create a remote SDR receiver that is available from the Internet, so that amateur radio operators can connect to the server with a web browser, click on an interesting signal on the waterfall display, and listen to the demodulated audio stream, without the need of purchasing any dedicated hardware to experiment with SDR (as shown on Figure 1).

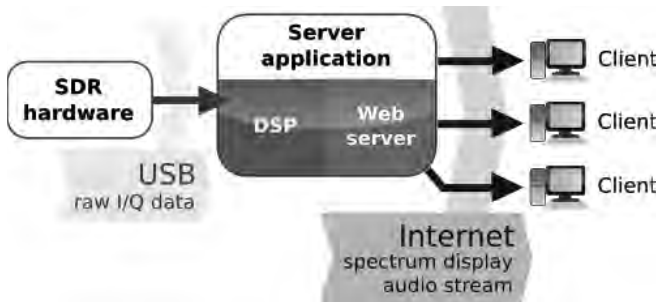


Fig. 1. A high level block diagram of OpenWebRX

II. ALTERNATIVES AND DESIGN CONSIDERATIONS

The idea is not new: there are several software available for this purpose. The best known is WebSDR, by Pieter-Tjerk de Boer, PA3FWM. WebSDR has been developed since 2007, and now has numerous great features, like the auto-notch filter and the HTML5 mode which also works on mobile devices. The drawback is the availability of

the software: as stated in the WebSDR FAQ, you can only get it directly from the author, so it is not distributed to anyone by means of free download, and the source code is not available at all, so even if you have the binaries, you cannot make your own modifications. Another good alternative is ShinySDR by Kevin Reid, AG6YO, which is open-source under the GPL license, but currently more suitable for use by only a single person at the same time (Kevin noted that multi-user support will be available in the future). One more notable project is WebRadio by Mike Stirling, which is not actively developed at the moment.

When I started working on OpenWebRX, only WebSDR existed as an alternative. I wanted to create a software package that implements an SDR receiver that satisfies the following requirements:

- it has a web UI (shown on Figure 2),
- it works with RTL-SDR dongles,
- multiple people can use it at the same time, and they can tune to different frequencies independent to each other, within the receiver bandwidth,
- the source code is available under an open-source license, as I consider reviewing, modifying and improving ham radio gear and software as a good practice.



Fig. 2. A screenshot of the OpenWebRX web UI

III. SETTING UP AN OPENWEBRX SERVER

To set up an OpenWebRX server at home, you will need a computer running Linux, and an RTL-SDR dongle. OpenWebRX has been tested on Debian/Ubuntu based systems, but it should work on other Linux distributions as well. If the dependencies have already been installed (*rtl-sdr*, *libfftw3-dev*, *git*, *python 2.7*, *gcc*, *bash*), then the first step to do is to get the git repositories:

```
git clone \  
https://github.com/simonyisk/openwebrx.git
```

```
git clone \  
https://github.com/simonyisk/csdr.git
```

Next, *libcsdr* should be compiled:

```
cd csdr  
make \  
sudo make install
```

Now OpenWebRX can be started. If it is ran with the default settings, it will set the RTL-SDR to the 2-meter band, and begin acquiring samples.

```
cd ../openwebrx  
./openwebrx.py
```

The receiver can be opened in the web browser by typing the following URL:

<http://localhost:8073/>

As OpenWebRX uses some recent HTML5 features that are not present in older browser versions, it requires the latest Google Chrome or Mozilla Firefox. It should also work on Chrome for Android, although it is not optimized for mobile devices yet.

By editing the *config_webrx.py* file, receiver settings can be configured (center frequency, sampling rate, gain) and also the details shown on the web UI (callsign, location, antenna, etc.) can be customized.

IV. USING OPENWEBRX

The user interface of OpenWebRX is similar to other SDR software: it has the appropriate buttons for changing modulation, receiver frequency can be set by clicking on a signal shown on the waterfall display. An additional feature is the availability of the whole history of the waterfall

display: one can go back in time with the help of the scrollbar on the right edge of the window. The filter bandwidth can be changed by dragging the ends of the yellow filter shape shown above the frequency scale (see Figure 3). Additionally, to achieve the same effect as turning the passband shift (PBS) or the beat frequency oscillator (BFO) knob on a real radio, the filter shape or the VFO tick should be dragged with the mouse, while holding the shift key down.

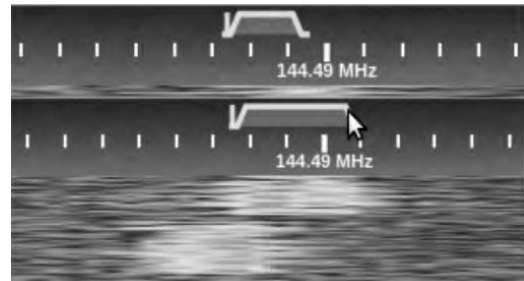


Fig. 3. Changing the filter bandwidth

V. PERFORMANCE NOTES

One of the greatest challenges in today's SDR receivers is the processing of the high amount of acquired data in real time. Even RTL-SDR is capable of bringing 2.4 Msps of 8-bit I/Q data into the computer, which means 4.8 megabytes of raw data to process every second. It is not a problem if you want to downconvert and demodulate a signal for a single user, but otherwise the CPU time required to process a block of data gets multiplied by the number of users. If too many clients are connected to an OpenWebRX server, and the CPU cannot handle the processing tasks for all of them, they will get lagging audio. Therefore it is important to optimize the settings to find the compromise between the high receiver bandwidth and the high number of users. There are several settings you can tune in *config_webrx.py*:

- *samp_rate*: The sampling rate directly affects the CPU usage per client. The lower it is, the more clients can be served simultaneously. For *rtl-sdr*, some valid values are 250000, 1024000, 2048000, 2400000.

- *fft_size*: The waterfall display is calculated only once for all clients, but decreasing its resolution can still improve performance (and results in lower network usage as well).
- *max_clients*: It is important to set this value to the safe maximum that will not result in lags.

You can stress-test OpenWebRX by opening your URL in a lot of tabs simultaneously, preferably on another computer. Note that doing this requires sufficient CPU performance at the client as well. On the server, you can check CPU usage with the *top* command.

You do not necessarily need a desktop PC to act as a server: you can also use one of the popular single-board computers (SBC) based on ARM processors. These cheap, small boards have low power consumption. It was reported by several users that the Raspberry Pi 2 is capable of running OpenWebRX, while the original, single-core Raspberry Pi is not sufficient. My tests on the Odroid-C1 board showed that it can serve at least 10 clients when the sampling rate is 240 ksp/s, so it would be all right for sound-card based SDRs as well. The audio started to lag when the number of clients exceeded 15.

On the other hand, running OpenWebRX on a quad-core Intel i7 CPU can serve 10 clients without problem at the RTL-SDR maximum sampling rate, 2.4 Msps, which is equivalent to processing 48 megabytes of data every second. This is not a bad result, regarding that we are talking about a general purpose processor. The CPU-critical process is the digital downconversion (DDC), which involves frequency translation and downsampling. Other DSP operations (like the actual demodulation and the AGC) work on a relatively low sample rate signal, so they do not take much CPU time.

VI. SYSTEM OVERVIEW

In the following part, the components of OpenWebRX are explained. The two main parts are the server application and the front-end

running on the client computer (shown on Figure 4).

The server has been implemented in *python*, a very flexible scripting language, which has definitely helped to cut down the time required for development. Along with running the web service, the server spawns several processes which communicate via OS pipes and TCP sockets with each other and the server core.

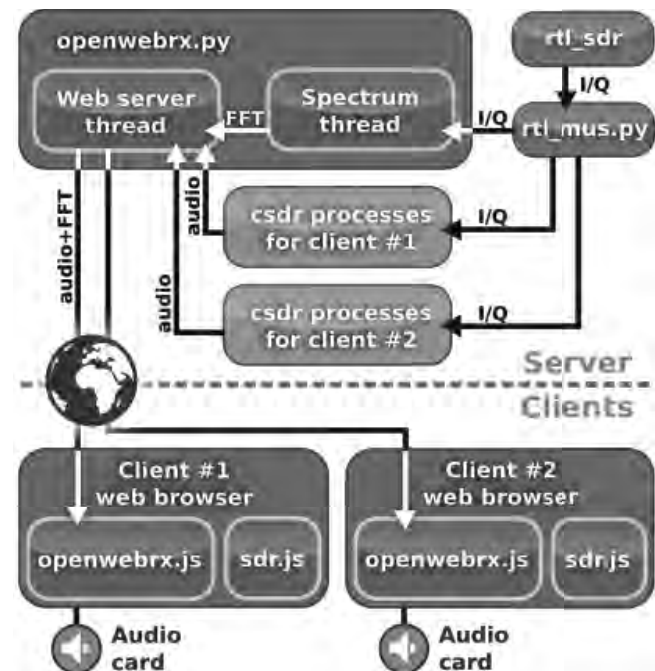


Fig. 4. A more detailed block diagram of the system

One of these processes is the well-known *rtl_sdr* command-line tool, which acquires the samples from the RTL-SDR device. In fact, it can be substituted with any other command that can emit I/Q samples to the standard output, so adding support for other SDR hardware should be easy, one just needs to change the command specified in *config_webrx.py*.

The component used for distributing I/Q data between processes is *rtl_mus.py*, which stands for RTL Multi-User Server. I released this previously as a separate open-source tool. It acts as a TCP server to stream raw I/Q data to multiple clients. (The version packaged with OpenWebRX has the

raw I/Q data port restricted to local connections only, for use by other server components.)

Every time a new client opens the webpage of the receiver, the browser initiates a WebSocket connection, which can be used for efficient two-way communication between the web server and the client (unlike traditional HTTP requests).

On the server side my own WebSocket implementation is used, which resides in *rxws.py*. When the WebSocket connection is established, the server spawns a set of new *csdr* processes as a signal processing chain, which takes its input from *rtl_mus.py*, and outputs the demodulated audio. The audio is then sent through the WebSocket by the server core, along with the spectrum data for the waterfall diagram. The spectrum data is emitted by the spectrum thread, which is common for all clients and thus has only one instance.

The front-end application running in the browser has been implemented in HTML5 and JavaScript: *openwebrx.js* manages the UI and the WebSocket, draws the waterfall diagram, and outputs the audio to the sound card using the Web Audio API. But before the audio can be output, some additional signal processing steps should be taken at the client side, which are performed by *sdr.js*.

VII. LIBCSDR

The DSP functions behind OpenWebRX have been implemented as a standalone library called *libcsdr*. There are already existing software packages for this: GNU Radio is the most notable, which is used by many SDR software for Linux, e.g. gqrx, ShinySDR. One of the reasons why I chose to implement my own lightweight solution is that compiling the latest version of GNU Radio from source takes a lot of time, and it is sometimes difficult to do. *libcsdr* contains only the required functions for AM/FM/SSB/CW demodulation, and should compile in a couple of seconds, even on SBCs.

I optimized the code for the auto-vectorization feature of the GNU C Compiler, so some DSP functions can make use of the SIMD instructions

in the CPU, although much more speedup could be achieved by hand written assembly code.

VIII. CSDR, A COMMAND-LINE TOOL FOR ANALOG DEMODULATION

Another feature of the DSP library is the *csdr* command-line tool, which can be used to build simple signal processing flow graphs with one-line Linux commands. This is the wrapper around *libcsdr* that OpenWebRX uses.

With *csdr*, prototyping a headless ham receiver on an ARM-based SBC might be quite straightforward. Example commands for demodulating NFM/AM/SSB can be found on the *csdr* GitHub page [3]. For the sake of simplicity I rather present a one-liner for demodulating a WFM broadcast at 100.2 MHz:

```
rtl_sdr -s 240000 -f 100200000 -g 20 - | \  
csdr convert_u8_f | \  
csdr fmdemod_quadri_cf | \  
csdr fractional_decimator_ff 5 | \  
csdr deemphasis_wfm_ff 48000 50e-6 | \  
csdr convert_f_i16 | \  
mplayer -cache 1024 -quiet -rawaudio \  
samplesize=2:channels=1:rate=48000 \  
-demuxer rawaudio -
```

In this example, to demonstrate the headless functionality, the raw I/Q data is taken from the output of the *rtl_sdr* tool, and at the end the demodulated audio is fed into *mplayer*, to play it on the sound card without the need of OpenWebRX.

We perform the following processing steps in separate processes:

- **rtl_sdr -s 240000 -f 100200000 -g 20 -**
RTL-SDR outputs I/Q samples with a sampling rate of 240 ksp/s.
- **csdr convert_u8_f**
First we convert the 8-bit unsigned samples to floating point.
- **csdr fmdemod_quadri_cf**

We run a quadricorrelator FM demodulator, which turns our complex input signal into a real output signal.

- `csdr fractional_decimator_ff 5`
We decimate the signal by a factor of 5, while also running a filter on it to suppress the possibly overlapping high frequency components. We get a 48000 sps signal at the output, which matches the sampling rate of our sound card.
- `csdr deemphasis_wfm_ff 48000 50e-6`
We apply a de-emphasis filter with a time constant of 50 μ s.
- `csdr convert_f_i16`
We convert the floating point real samples to 16-bit signed integers to match the format required by the sound card.
- `mplayer -cache 1024 -quiet -rawaudio (...)`
We output the samples to the sound card.

As the DSP functional blocks run in separate processes, the flow graph can take advantage of multiple CPU cores, if available. Scheduling is handled by the operating system. I was skeptic for the first time I have tried this, as I know the importance of scheduling in a dataflow system. While experimenting with OpenWebRX and *csdr* together, it turned out that this solution does quite good job even if about 50 processes are started when multiple users are present. In addition to multi-core processing, this implementation makes the flow graphs of OpenWebRX easily modifiable, as they are defined as a few lines in *plugins/csdr/plugin.py*, and also keeps *libcsdr* code simple.

IX. EXPERIMENTS WITH CLIENT-SIDE DSP

The first release of OpenWebRX server required about 2 Mbit/s uplink network bandwidth per client, because I sent the 16-bit 44100 Hz raw sample stream through the WebSocket. In order to make OpenWebRX suitable for hams who want to share their receiver from their home Internet connection with a low upload speed, I added two processing steps before sending the data over the network connection:

- I decreased the sampling rate to 11025 Hz, which is still enough for NFM and SSB transmissions.
- I applied IMA ADPCM compression to the audio.

As a result, the required uplink bandwidth now is about 200 kbit/s with the default settings, of which about 70 kbit/s is the audio, and 130 kbit/s is the spectrum data.

In order to restore the original signal at the client side, I had to implement the reverse operations in JavaScript.

JavaScript has developed much in the last few years. It was traditionally an interpreted language, which resulted in with very poor execution speed, but with the introduction of JIT compilers, namely TraceMonkey in Mozilla Firefox and the V8 Engine in Google Chrome, JavaScript is now feasible to build complex applications on.

Some of the new, and still not widely known achievements on the scene of JavaScript are the Emscripten compiler and *asm.js*. The Mozilla Foundation has designed *asm.js* to be a subset of JavaScript that JIT compilers can easily optimize for, so performance is typically 50-67% compared to the speed of the native version of the same code [4]. Emscripten is an LLVM-based, source-to-source compiler to translate C/C++ applications into the *asm.js* subset of JavaScript. For example, companies use it to port games to the web, by compiling the original C++ code of the game (that was linked against OpenGL) into JavaScript code that uses WebGL, so the game can be played directly in the browser.

Instead of porting some algorithms from *libcsdr* from C to JavaScript manually, I decided to compile the entire *libcsdr* package to JavaScript with Emscripten. The resulting JavaScript library has been called *sdr.js*, and contains all the functions that *libcsdr* has. However, there are some drawbacks. First, a lot of additional wrapper code had to be written to actually use the compiled functions. At this point only functions that are essential to OpenWebRX (like the IMA ADPCM codec and the resampler) have wrappers yet.

It is important to note that here I do only some post-processing on a signal with relatively low sample rate (<50 ksp/s), so the 50% performance difference between JavaScript and the native code is not a problem. However, if someone would want to write a standalone SDR application running in the browser, the speed of the DDC operation and the FFT calculation would be limited compared to a native implementation, and also SDR hardware access is complicated from the browser (although Google's Radio Receiver application for Google Chrome, which has entirely been implemented in JavaScript, works in a similar way).

Back to the compression, it is also worth to mention that the spectrum data is compressed, too. As it is a one-dimensional vector of dB values, it was not feasible to use an image compression algorithm like JPEG. I've had the idea to test whether ADPCM works on it, as the spectrum data is just like any other real-valued signal, and not that different from an array of audio samples. Surprisingly, I had good results, so I kept using it.

X. REAL-WORLD USES OF OPENWEBRX

OpenWebRX is currently used by only a few stations.

Richard van der Riet, PA3GWH and Bart Weerstand, PA3HEA have been running their servers since February. Richard's server is monitoring the 30 meter band, while Bart's is currently working on 2 meters.

With the help of Levente Dudás, HA7WEN I had the chance to do some testing on one of the machines of HA5MRC Radio Club in Budapest, with a receiver connected to a Yagi antenna array automatically pointed to the JAS-2 amateur satellite during its pass.

Antal Vincze, HG4FC was the first to test the OpenWebRX server in production environment at HA5KAW club station in Nadap, Hungary.

John Seamons, ZL/KF6VO has adapted OpenWebRX for use with his direct sampling HF receiver.

Alex Trushkin, 4Z5LV has made a version of OpenWebRX that is compatible with AFEDRI SDR.

A few days before submitting the paper I have got the news that the first OpenWebRX server on HAMNET has become available, by Hans Reiser, DL9RDZ, and there is one more to be set up soon.

XI. CONCLUSION AND FUTURE PLANS

Today's technology allows us to implement complex applications like an SDR receiver as a web service. A multi-user SDR receiver requires much more CPU performance than a single-user application, still we can find a compromise to let the application run on single board ARM computers.

As young people of today have born into a world with computers and mobile phones, I hope that web receivers will make some of them interested in the great hobby of amateur radio.

If you are further interested in the OpenWebRX project, you can find more information at the following website:

<http://openwebrx.org/>

On this page, my Bachelor's thesis that I have written on this topic is also available for download. It contains more details about the implementation of both the web service and the signal processing. I hope it will help those who want to write the code for their own ham radio SDR receiver.

The development has not stopped after submitting my thesis. The website for listing the receivers is expected to be ready soon. Users have requested many improvements, for example a squelch, and support for tablets, so there is a lot of things to do. The DDC could be much more optimized in order to be able to serve more clients from the same host, and using GPU hardware acceleration is also under consideration.

XII. ACKNOWLEDGEMENTS

I would like to thank to everybody who helped me with this project, and especially my friends János Selmeczi, PhD (HA5FT) and Péter Horváth, PhD (HA5CQA) for their continuous help and support.

REFERENCES

- [1] OSMOCOM (2015. 07. 15.), *rtl-sdr*. Available: <http://sdr.osmocom.org/trac/wiki/rtl-sdr>
- [2] András Retzler (2015. 07. 15.), *SDRLab: an RTL-SDR interface to LabVIEW for educational purposes*. Available: <http://ha5kf.u.sch.bme.hu/sdr/lab>
- [3] András Retzler (2015. 08. 16.), *csdr*. Available: <https://github.com/simonyiszk/csdr>
- [4] Alon Zakai (2015. 08. 16.), *Emscripten & Asm.js: C++'s Role in the Modern Web* (slide 26). Available: https://kripken.github.io/mloc_emscripten_talk/cppcon.html#/26

Modulation – Demodulation Software Radio

Yahoo user group: <https://groups.yahoo.com/neo/groups/mdsradio/info>

MDSR website: <http://users.skynet.be/myspace/mdsr>

Build your own IF SDR Introduction of MDSR V3.0

Alex Schwarz, VE7DXW; alexschwarz@telus.net

Guy Roels, ON6MU; guy.on6mu@skynet.be

Accomplishments since the last publication

Base Station setup

The MDSR test station currently in use is a modified FT-950 from Yaesu. The transceiver is connected to a Hy-Gain 18HTjr vertical antenna on top of a 3-storey building. Ground is connected to an iron riser pipe. The station is located on a foothill 10 miles north of Vancouver. Power output is 20W for digital modes and 100W for voice.

The computer running the MDSR software is an i5 processor running at 3.4GHz with 8GB of RAM. The installed operating system is either 64bit Win 8.1 or Ubuntu 14. The sound card installed is an ASUS Xonar Essence true 24bit Performance. The on-board sound device is a Realtek High Definition on-board audio device.



Mobile setup



The mobile setup is a modified IC-7000. The antenna is a telescoping HF antenna from Hy-Gain (MFJ-1979) with homebrew loading coils for 40 and 80m (left); used for stationary operations. A 4-band antenna (CR-8900 - 10m, 6m, 2m, 70cm) is used while driving (right). The home made antenna rack fits multiple vehicles and can be transferred from one vehicle to the other in minutes. The rack includes grounding rods that can be connected on each end for better performance in stationary operation.



Mobile setup of MDSR in a Honda Element



The IC-7000 radio is mounted in the lockable glove compartment which is closed while the car is parked or the radio is not in use (computer and BiLIF are packed away in a box).

The silver box on the dash contains the BiLIF unit and the Xonar U7 USB soundcard, made by ASUS.

The computer running the MDSR software is an ASUS Transformer T100 book running Win 8.1 32bit. It also runs the digital communication programs for JT-65 and WSPR which use the internal sound card.

A USB hub connects all the devices together. The IF and the PTT connection come from the transceiver and plug into the BiLIF.

Note: The IC-7000 does not need a control head – all functions are performed by the MDSR software via CAT control. The IC-7000 control head is only used during mobile operations.

Portable Setup

The portable setup consists out of a modified FT-817, a silver box that contains the BiLIF and the USB sound device. The mic/headset plug connects to the web-book internal sound card for digital operation.



The Portable MDSR system packed away in 2 travel cases for easy transportation

On the left (smaller case) the antenna utility box contains all various antenna parts, such as loading coils, antenna extenders and the mounting bracket. The long rod across both cases is the MFJ-1979 telescopic antenna which extends to 5m (15ft.). It does not fit into the cases and has to be carried separately.

In front of the small case is the Asus T-100 Transformer book, which runs the MDSR software.



The big case contains all the wire connectors, chargers, power adaptors, hub, spare batteries (8 x 2500mA/h NiMH) and the antenna cable (hidden behind the lid cover) and the BiLIF-U7 sound device box (silver). The ASUS T-100 can be wrapped in towels and laid on top of the FT-817 for transportation. The standard NiCad battery pack in the FT-817 has been replaced with 8 2500mAh NiMH rechargeables. The battery pack has been modified to allow charging of the batteries without removing them from the radio.

The whole setup weighs less than 15lb. One charge lasts for about 4h of operation (TX output reduced to 2.5W). The batteries can either be charged from the mains or with 12V from a cigarette lighter adaptor.

Awards received and in progress

- WAS50 in mixed bands for JT-65 mode
- Worked all continents of the world in mixed bands for JT-65 mode
- 100 Grid Squares in mixed bands for JT-65 mode
- Worked 53 states (confirmed 44 on e-qsl and 31 on QRZ.com) as of June 2015

Hardware

The LIF2014 kit was released. The schematics and the components are identical to the previous LIF2012 kit that has worked so reliably. The only update was the footprint of L1 to make it easier to assemble the unit.

A lot of Hams have already built and implemented the kit into their station setup; they use it every day for standard operations.

Software

The MDSR Team has released the MDSR V2.9 last fall. This release added notch filters and adaptive filters for SSB, CW and broadcast audio. The graphic user interface was also changed to allow for better navigation and to reduce the footprint on the screen. If the MDSR is used only for RX operation the window size can be reduced to only show the button and display for this task.

The post-processing filter options

Depending on the mode the MDSR is in, different filter options are displayed. This allows the optimization of each post-demodulation filter for better overall performance of the MDSR while receiving. In addition to a variable bandpass filter, one auto notch and one manual notch filter can be utilized to eliminate interfering signals.

Note: AM mode does not have an auto notch filter.

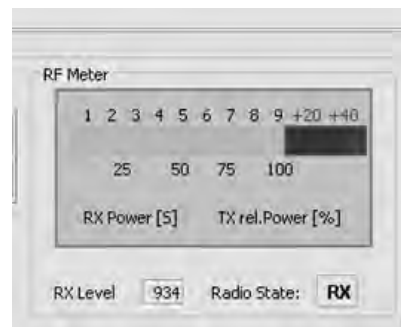


Version 3.0 was launched in February 2015. It received a totally new DSP engine to upgrade to the Jsyn 100% Java sound synthesizer. Now the MDSR core software can be adapted to all Java capable computers such as Linux, Ubuntu and Apple.

The MDSR V3.0 can be downloaded from our website and used for free for amateur radio purposes. For more info please visit our website or join the Yahoo user group.

New S- and Power Meter for the MDSR V3.0

The updated RF meter features a dual color display for RX. It turns blue while in TX mode. In receive the scale is in S units, and the meter displays Pout [%] during transmit.



The new “MDSR SA” Spectrum Analyzer

Introduction

Implementing the latest Jsyn DSP software in the MDSR V3 also gave us access to advanced signal processing devices such as FFT and IFFT conversion units. Our team started to develop a spectrum analyzer by using a spectral filter. This filter creates a 420 line audio spectrum (0 – 20 kHz) with a resolution of about 47Hz per line. After the first attempt to connect the SA to the IF of the LIF converter the result was encouraging, but the bandwidth response of the receiver’s IF looked like a “hump”. This made the detection of signals very cumbersome and useless for level comparisons.

The idea was born to compensate each spectral line individually so that the result was a flat spectrum. This is not practical, though, because it would take the user hours to adjust all 420 line levels individually. Sweeping the bandpass was another option, but this would require a RF sweep generator. Most amateur radio operators do not have access to such an instrument. Another option was to measure the receiver’s filter responses with RF noise, by tuning the radio to an unused section of the band and then taking a noise peak measurement for each frequency line.

The noise that is received on the antenna port is constant over a wide part of the spectrum. Therefore all the level differences are caused by the filter pass of the input and IF filters. So what looks like a “hump” – when noise is received – is really a flat spectrum. Now the DSP could be programmed to measure the peak level of each spectral line, calculate its attenuation and store the result in a table.

To obtain a flat spectral readout now only requires a simple multiplication of the corresponding attenuation factor by the current spectral input level. The result is a displayed spectrum that is flat. In essence the effects of the receive filtering, which are important for the radio's performance, have been mathematically removed.

How do Spectral Filters work?

Introduction

Spectral filters are a very interesting way to manipulate and create signals. The standard oscilloscope only measures voltages as the sum of electron pressure in conducting materials over time. It does not provide a frequency-separated display that shows the dynamic behavior of all the different groups of electrons that make up the electric current. These groups of electrons can move back and forth at different speeds (frequencies). The more electrons move in one direction, the higher the amplitude. Frequencies or tones are in essence oscillating electrons.

If tones are to be manipulated, it is much easier if the signal is separated by frequency (tones) than by time. Unfortunately it is much harder to display electrical behavior of a conductor in the frequency domain than in the time domain of an oscilloscope. This is also true in DSP (digital signal processing).

What is FFT (Fast Fourier transform)?

In DSP the frequency domain is so important that engineers developed special programs called FFTs that take streaming audio or video data and separate it into multi-channel data flows. FFTs can have 1000 or more channels separated by frequency; these are termed *bins*. These individual data flows can now be observed by frequency and amplitude for a specific behavior. A host program can display the data as a spectrum on a screen or make decisions if certain signal patterns are present. The applications of this technology in the radio field alone are almost limitless.

What is IFFT (Inverse Fast Fourier transform)?

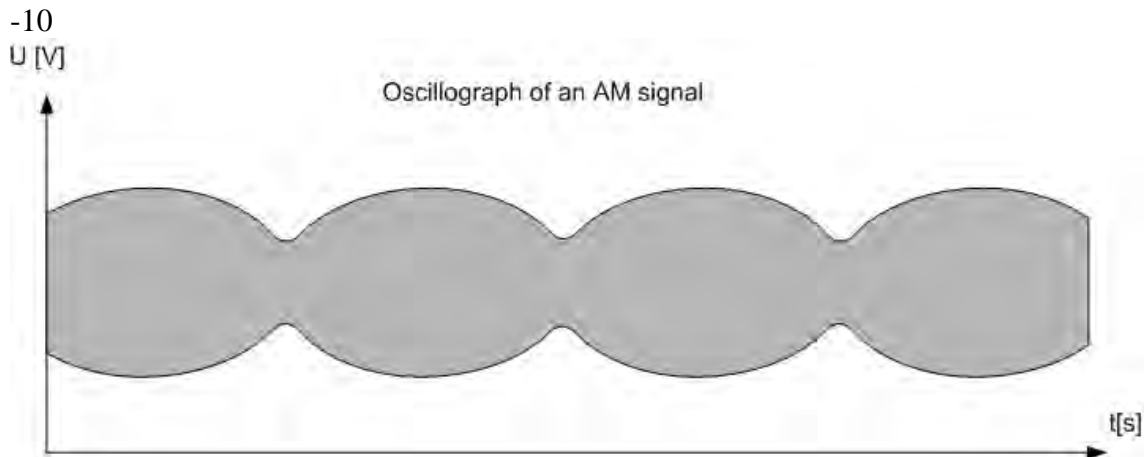
In DSP, IFFTs has an almost equally important role. IFFT and FFT are mostly used in pairs. Whereas FFT separates the input signal into different frequency channels the IFFT does exactly the opposite. It takes all the FFT bins – after they were processed – and combines them together into a time domain output that can drive a speaker after it has been turned back into an analog signal by a DAC (Digital to Analog Converter).

Applications

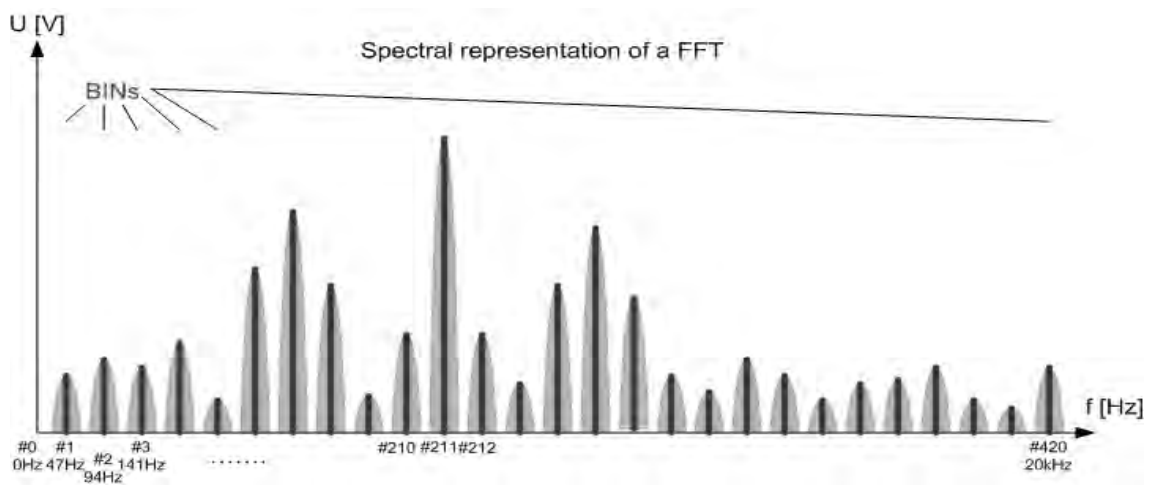
- Removing unwanted frequency bands
- Shifting frequencies or bands of frequencies up and down
- Removing unwanted carriers or static crackle
- Modulation and demodulation (digital or analog)
- Adding control tones
- Creating noise

Spectral Filter example with FFT / IFFT pair of 420 channels (BINS)

An AM signal is fed to the ADC and changed into a stream of numbers relating to amplitude.



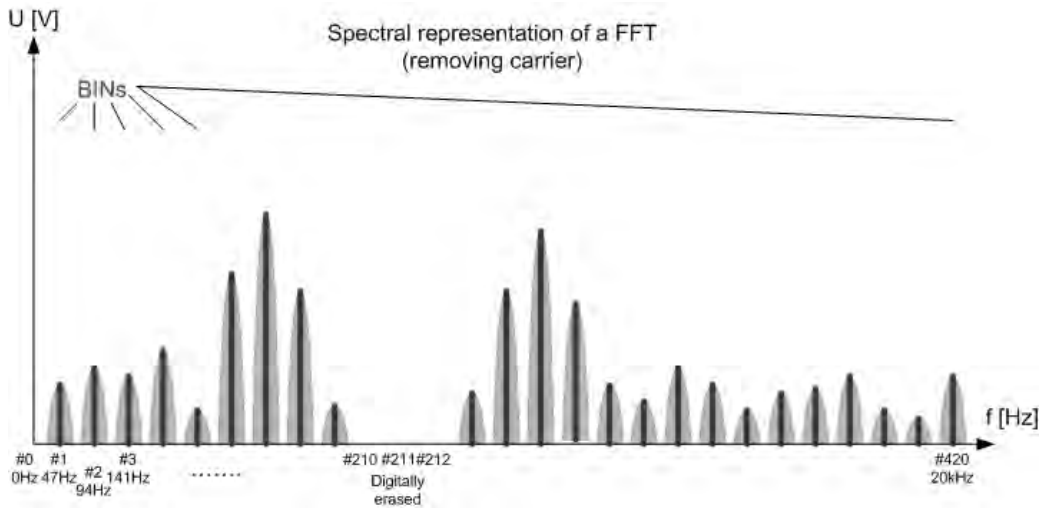
The now digitized signal is put through a DSP process called FFT



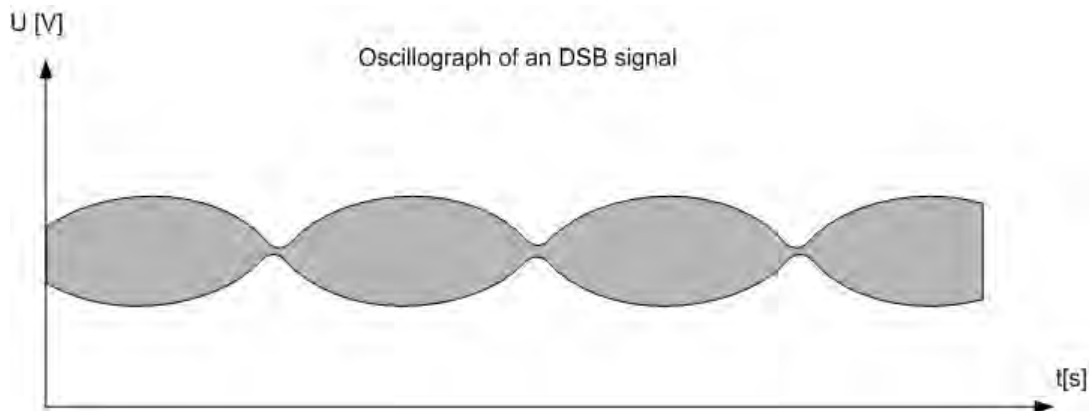
The FFT filters frequencies and stores the signal stream in BINS (table representation)

| time \ Bin # | 0 | 1 | 2 | 3 | 4 | 5 | 6 | ... | | | | | | | | | | | | 420 |
|--------------|----|----|----|----|----|----|----|-----|--|--|--|--|--|--|--|--|--|--|--|------|
| 000154 | a0 | a1 | a2 | a3 | a4 | a5 | a6 | ... | | | | | | | | | | | | a420 |
| 000155 | a0 | a1 | a2 | a3 | a4 | a5 | a6 | ... | | | | | | | | | | | | a420 |
| 000156 | a0 | a1 | a2 | a3 | a4 | a5 | a6 | ... | | | | | | | | | | | | a420 |
| 000157 | a0 | a1 | a2 | a3 | a4 | a5 | a6 | ... | | | | | | | | | | | | a420 |

Signal manipulation based on frequency (removing Bin 210 - 212)



Performing an IFFT (Inverse Fast Fourier Transform) and sending the data stream to a DAC will recreate the analog signal without the carrier.



Disadvantage of FFT filters

- All the data manipulation is processor intensive and delays are an issue. The program needs to reduce the amount of DSP data to the bare minimum before executing DSP functions. The software instructions need to be as basic as possible. Avoid coding that requires access to higher functions or peripherals such as keyboard, mouse or displays.
- The DSP processes can create huge amount of data. A 24-bit DSP chip with an FFT having 1000 channels and a sample rate of 192kHz creates 4.608MB of data every second!
- A fast computer is required to get the speed performance needed. Good coding can assist in reducing processor requirements.
- Signals can sound tinny and robotic due to the FFT- IFFT process. A good understanding of how to configure the DSP properly is a must.
- The high price of components and the availability of high resolution digital converters are limiting factors.

MDSR SA

Introduction

The new spectrum analyzer is written 100% in Java – including the DSP – and enhances the MDSR software on a Linux platform. Since it is written in Java it also will work with the existing MDSR installation in Windows.

One of the issues with viewing an IF spectrum from an analog transceiver via a sound card interface is the inconsistency of the IF amplitude and the narrow bandwidth. This makes the spectrum look like a hump with no signals on either side. To overcome this we have developed anew feature called SAC (spectral amplitude correction) which measures the attenuation of the IF and applies a correction value that greatly flattens the spectrum.

This document will also explain how to install and run the software in the above mentioned operating systems. A detailed manual is also provided.

This is a working document which will grow in size as we explore the possibilities of this new software. If you would like to join us and work on the continuing project, please register at the

MDSR Yahoo user group here:

<https://groups.yahoo.com/neo/groups/mdsradio/info>

Information on how to connect a receiver or transmitter to the soundcard can also be found at our website hosted by Guy Roels (ON6MU):

<http://users.skynet.be/myspace/mdsr>

Acknowledgement:

Special thanks go to Phil Burk for providing the Jsyn FFT sound interface free of charge for this project.

Installing on Windows with MDSR already installed.

Download the “Spectrum Analyzer. jar” from the Yahoo use group. Look in the file section in the “New SA for MDSR” folder. Copy the file into “MDSR” directory. Create a desktop shortcut. The program is started by double clicking the file or shortcut.

Note: If the spectrum analyzer is used without the MDSR, additional library files are required. Create a “lib” folder inside the folder holding the “Spectrum Analyzer. jar”. This “lib” folder and its contents can also be downloaded from the group.

Installing on Ubuntu (Linux), Apple Computer

Download the “Spectrum Analyzer. jar” from the Yahoo user group. Look in the file section in the “New SA for MDSR” folder. Copy the file into MDSR directory. Create a desktop shortcut. The program is started by double clicking the file or shortcut.

- Make the file executable by going into properties – permissions and checking the “Allow executing file as program”

Note: If the spectrum analyzer is used without the MDSR, additional library files are required.

Create a “lib” folder inside the folder holding the “Spectrum Analyzer. jar”. This “lib” folder can also be downloaded from the group. It contains three files that are required to make the program work.

Press the “Start” button to turn the spectrum analyzer on

Setting up the Soundcard or Device

The sound setup of the “New MDSR Spectrum Analyzer” is identical to the setup of the MDSR software. It always uses the default audio device, and can share it with other software at the same time. For more information on how to install and configure the LIF converter, go to the MDSR website or user group. If you have already set up the MDSR software, all the audio settings are identical. Both programs can work simultaneously without mutual interference.

Functionality of the Spectrum Analyzer

Introduction

This spectrum analyzer provides a span of spectrum for the audio spectrum of a sound card. It is designed to work with the MDSR software, but can also be useful for analyzing audio spectrum. It has a peak hold feature and a slow decay indicator.

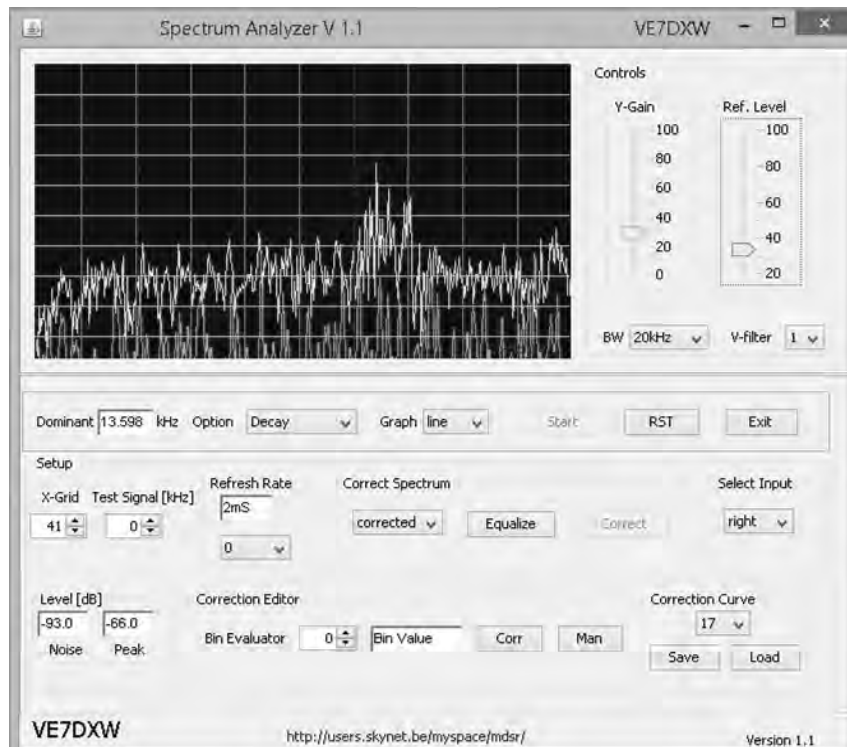
The new SAC (spectral amplitude correction) feature and how to use it will also be covered in this manual.

Updated in the new version V1.1

- **New line decay mode:** double spectral lines in “decay-line mode”.
- **Input Audio selector:** allows the user to select the left, right or both input audio channel
- **Storable correction curve files:** correction curves can be stored according to band
- **Spectrum analyzer stores previous settings:** for easy startup without calibration
- **Updated graphic user interface**

The new MDSR Spectrum Analyzer V1.1

Below is the graphic user interface for the new MDSR spectrum analyzer with the corrected display. The DSP removes the filter skirts which cause the spectrum to appear bent downwards at the edge of the filter passband. A flat spectral trace – like the one on a high-end lab spectrum analyzer – is displayed.



Correcting the filter skirts to obtain a flat spectral response

Correction Procedure

The correction procedure is a two step process where the filter noise response is captured and stored in the computer. Once the curve had been captured, it is applied to the spectrum and a normalized spectral display results. The filter noise response depends on the noise loading and the band the transceiver is on. The noise response can be saved and recalled for each band separately.

How to calibrate and correct the filter response in the MDSR SA

Depending on the speed of the computer used, this calibration will take about 10s or longer.

1. Tune the receiver on the specific band in use to an area that has no signals. Set the attenuators to 0dB and the preamps to OFF.
2. Under the "Corrected Spectrum" heading set the selector box to "normal". The spectrum changes back to the real display with the filter skirts (Fig. a).
3. Press the "Equalize" button and a display similar as seen below appears (Fig. b). The white dots indicate the flat response of the corrected spectrum. The green line represents the correction values. If the green line is not visible, move the "y-Gain" and/or the "Ref.Level" controls until it becomes visible. If there are signals present, they will be seen as large spikes on the filter curve. Tune to another spot without signals and repeat the measurement starting at 1. Small spikes, like the ones below, will be filtered out and a flat spectrum is displayed.
4. Press the "Correct" button to verify the correction line and to display the corrected flat display. Use the "y-Gain" and/or the "Ref.Level" control to adjust for best visibility.
5. If the spectral line is not flat, repeat the process.

Note: The measurement must be done with the antenna connected, so as to obtain band noise to measure the filter response.

Calibrate and correct the filter response in the MDSR SA using a white noise generator

The spectrum display correction curves can also be generated with the use of a wide band white noise generator, such as the MFJ-5014. To get the best results, the antenna noise level and the generated noise level have to be identical.

The output of the generator is connected to a step attenuator such as the MFJ-762 and then to the antenna input of the receiver. If the receiver has dual Antenna inputs, the RF noise can be connected to Ant 2, while the regular antenna is connected to Ant 1.

1. Connect the Antenna to the receiver and set the attenuators to 0dB and the preamp to OFF (Ant 1).
2. At the receiver, select the band to be measured.
3. In the "Level [dB]" box (bottom left) note the displayed level in the "noise" field.
4. Connect the receiver to the noise source (Ant 2).
5. Adjust the attenuator so that the displayed noise level is the same as that from the antenna.
6. Continue the above procedure at #2.

How to store the filter response in the MDSR SA for later recall

Once a flat filter response correction curve has been calculated and the corrected spectrum is flat, it can be stored on the computer's hard drive. Since the flatness of the display is dependent on the band and the noise loading, the MDSR SA has the capability to save one correction curve per band. Currently the MDSR SA stores curves for 160, 80, 60, 40, 20, 17, 15, 12 and 10 meters. After a flat spectrum has been calculated, it can be archived by following the above procedure for a specific band:

1. Under the "Correction Curve" heading, select the band that is currently in use.
2. Press the "Save" button. This will store a file containing the correction values.
3. Repeat the above procedure until all bands have a correction curve file.

How to recall the filter response in the MDSR SA

1. Under the “Correction Curve” heading select the band that is currently in use.
2. Press the “Load” button. This will recall a file containing the correction values.
3. The displayed spectrum is now corrected and should be flat.

Note: If the selected band does not have a correction curve, a warning message will appear.

Fig. a.) Spectral Display without correction (line mode)

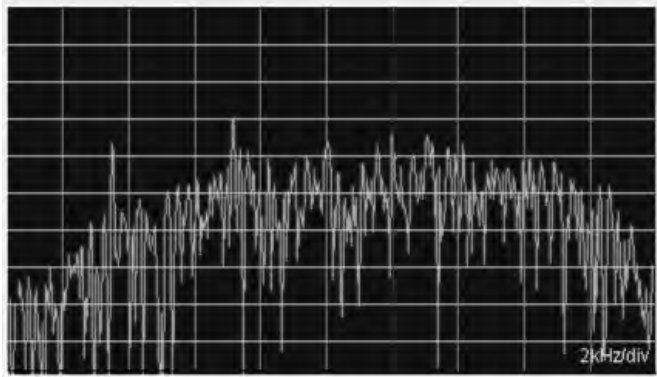


Fig. b.) Spectral Display while calibrating (line mode)

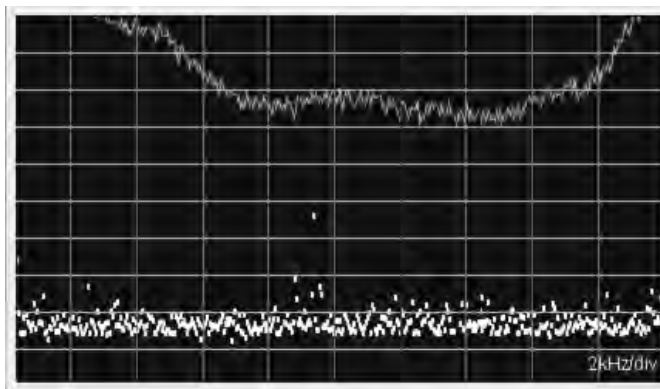
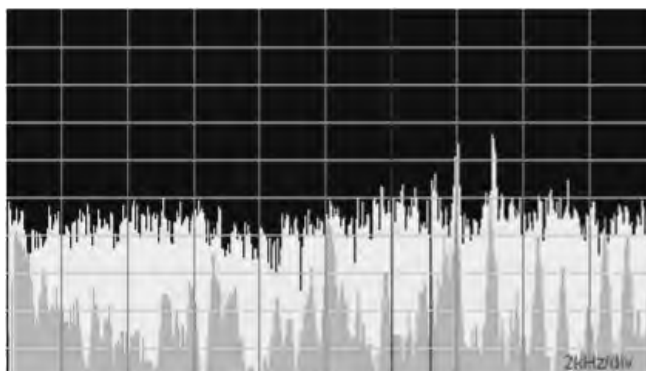


Fig. c.) Spectral Display after correction (bar graph – decay option)



Description of the user functions.

Introduction

This section describes the all the individual user controls and how they affect the displays and functionality of the MDSR SA software. Familiarization with their use and effects will allow you to adjust the display properly and to get the most out of this program.

Start Button

The “Start” button initializes the DSP and turns the spectrum analyzer on.

RST button

During operation, some sound devices build up a delay over time. This occurs, because the clocks of the FIFO data exchange buffers are not synchronized (a hardware issue). If there is noticeable delay, the “RST” button can be pressed to reset and clear audio buffers for lowest lag time.

Exit Button

The “Exit” button stops and removes the DSP engine from the computer and closes the program.

Dominant

The “Dominant” field displays the frequency value of the spectral line with the highest value.

When the spectrum is corrected, the highest amplitude may not be the dominant frequency.

Option (selector box)

The “Option” selection box has three options.

- Normal: (default) displays the spectrum as a single graph.
- Peak hold: displays the graph including its highest value. The highest value is held on the display indicating its frequency position.
- Decay: displays the normal graph including a time-lagged, delayed decay graph. This makes it easier to spot intermittent carriers.

Graph (selector box)

The “Graph” selection box allows choosing the line display or the bar graph display. The “line” setting is the default start up parameter.

Y-Gain

The “Y-Gain” slider changes the amplification of the displayed spectrum. It acts like a volume control and makes the spectral peaks longer or shorter depending on the setting.

Ref. Level

The “Ref. Level” control shifts the displayed spectrum up and down so that it can be placed properly in the grid display for easy readability. The control acts similarly to a fader on a stereo system. “Ref. Level” does not change the amplification of the display, but only its position.

BW (selector box)

The “BW” selection box provides three settings

- 20 kHz (default) provides a spectral display with 420 points at a resolution of 47 Hz. For the MDSR or IF display the 20 kHz selection has to be used to allow for proper placement of the red BFO line. The displayed spectrum is 0 – 20 kHz.
- The 10 kHz option can be used for audio processing and analysis. The spectrum displayed is 0 – 10 kHz. It will not center on the BFO line on the IF signal. It also has 420 display points with a spectral resolution of about 23 Hz.
- The 5 kHz option can be used for audio processing and analysis. The spectrum displayed is 0 – 5 kHz. It will not center on the BFO line on the IF signal. It also has 420 display points with a spectral resolution of about 12 Hz.

V-Filter (selector box)

The “V-Filter” (video filter) processes the spectral data and acts like a low pass filter to make the displayed spectrum smoother. There are five levels of filter intensity; 1 is the weakest and 5 the strongest. In the lowest setting (1) all the individual spectral lines are displayed. In the strongest setting the display lines are combined making it easier to see the spectral intensity of a SSB or broadcast signal. Default setting is 3.

X-Grid (spinner selector)

The “X-Grid” selector allows calibrating the spectral display with an internal or external signal source. The spectral display and the grid do not automatically line up, so it needs to be calibrated manually.

- Calibrating using the internal reference signal:
Set BW to 20 kHz, and turn on the “Test Signal” generator by setting the spinner control to 10. This puts out a white noise signal with sine wave peak at 10 kHz. Now, use using the “X-Grid” control, move the spectral peak behind the red center line. This will also calibrate the 5 kHz and the 10 kHz bandwidth selection.
- Calibrating using an external reference signal
The “X-Grid” control can also be used to be aligned with an external signal such as the carrier of an AM station, the radio’s calibration marker or a BFO. In this case the computer soundcard needs to be connected to a LIF converter and a receiver.

Test Signal [kHz]

The “Test Signal” selector allows adding an internal noise source and sine wave oscillator to the input of the spectrum analyzer. The test signal can be used to simulate the input for display or calibration purposes. Setting X-Grid (spinner selector) to 0 will turn the test signal off. Now the DSP is set to accept input from the audio default device.

Refresh Time

The “Refresh Time” default setting is “0”. This setting is the fastest, and if a P4 or higher processor is used the spectrum refresh rate is high enough to provide a nice, dynamically flowing spectral display. The FFT process needs a lot of processor time, and it can easily consume 60% CPU resources or more. This may affect other applications, and therefore a refresh time throttle is needed. Set the refresh time lower if other applications start to hang or are otherwise affected during the operation of the spectrum analyzer program. The refresh time indicator displays the time in mS for each sweep. Newer processors can be as fast as 1 to 2 mS.

Equalization (selector box)

The “Equalization” selector has three options.

- normal: the displayed spectrum is not corrected
- corrected: the display is normalized using the loaded correction curve.
- calibrate: this is used only during the calibration procedure – do not select manually.

Equalize button

The “Equalize” button should only be pressed after the receiver has been tuned to a section of the band without signals. During this operation, the DSP measures the filter response curve which can be viewed on the display.

Correct button

The “Correct” button activates the currently active (loaded) correction curve and corrects the spectral display.

Select Input

The input selector chooses the input channel of the default sound card. Three options are available (both, left, right).

Level [dB]

Under the “Level” label there are two fields that display the average noise level and the peak level per scan in dB relative to 1V_{pp}. These two variables are used to determine the bin with the highest amplitude, so as to calculate the dominant frequency value. The noise level indicator is also used for correction curve measurements.

Bin Value Evaluator

The "Bin Value Evaluator" consists of the bin value field and the bin number selector. The bin value field displays the correction value of the bin selected. This allows the user to view and edit stored correction values.

Note: A red marker bar is displayed in bar graph mode indicating the position of the displayed correction value.

Corr Button

The "Corr" (correct) button calculates the current displayed bin correction value by taking the adjacent values and averaging its position. This can fix miscalculated values which can cause holes or peaks in the spectrum.

Man Button

The "Man" (manual) button updates the manual entered correction value for the displayed bin.

Correction Curve

In the correction curve box, the option box selects the band that either saves or loads the correction curve data into the DSP. The "Save" button save the currently active correction data. To recall correction data, the "Load" button is pressed. The display is updated immediately.

Congratulations!

Now the new Spectrum analyzer for the MDSR is working and calibrated. Please follow the instructions and tune the new MDSR SA for best performance. We hope that you will have a great experience with the new spectrum display. If you have any questions or recommendations regarding this software or would like to get involved, please join us at the Yahoo user group:

<http://groups.yahoo.com/group/mdsr/>

Thank you for using the MDSR. Please tell your friends about this project.

All the best,

MDSR Team

June 2015

Design of a Practical Handheld Software Radio: Part II

Chris Testa, KD2BMH
Los Angeles, CA
testac@gmail.com

October 9, 2015

Abstract

The design of a standalone battery powered Software Defined Radio (SDR) is presented. Three rounds of prototypes were designed, built, and tested over the last three years. The hardware architecture of the newest design is detailed, with the goal of getting the device into the field to build real RF links. The software stack, from the high-level websocket user interface down to the embedded Linux operating system are discussed. Finally, the latest work on the Field Programmable Gate Array (FPGA) modem are presented, including optimization work that drastically improves simulation performance.

Keywords

software radio, low-power, embedded systems, Linux, FPGA, DSP, quadrature transceivers, RF system analysis

1 Introduction

In 2012 I reported my first success along the way of developing a new type of Software Defined Radio[1]; one in which the whole SDR is contained in a single, portable unit. I was especially inspired to do this for my love of backpacking, and I continually find myself in the position where I really want to have radio communications available in places where today you can't expect them.

The dream, as Eric Blossom wrote in the article Exploring GNU Radio[2], is to stretch the "smarts" of the Internet out from the cell towers to everyone's smartphone. The belief which I still hold today, is that if we all carry around a base station, we will be well on our way to distributed and fault tolerant Internet access worldwide. I know that this is a lofty goal, but with the right tools we can begin to explore new frontiers in networking.

My key goals for the project are to:

- Design and build a standalone, software defined transceiver that works with commonly available Amateur radio modes.
- Make it easy to use by providing connectivity and extensibility layered on top of Open Source software.
- Focus on small footprint and low power consumption to enable portable operation, much like with a cellular modem chipset.

At the time I presented at the DCC 2012, I had never designed or laid out a radio circuit board before. I studied Computer Engineering at the University of Maryland, College Park, and I've loved ripping apart computers from a young age. Radio Frequency circuits are an entirely different beast, however. There's a huge learning curve to building a SDR from scratch, and the remainder of this paper will detail the evolution of the design, and the things which I learned along the way.

2 Hardware

2.1 Design Evolution

The first pre-alpha design was completed thanks to the WIESEL laboratory at the University of Utah, and with the help of my good friend Aaron Schulman. Aaron at the time was finishing his PhD in Computer Science at University of Maryland. I built the first transceiver by ordering development kits for all of the main integrated circuits I wanted to use: a SoC FPGA¹, a quadrature transceiver, a frequency agile VCO & PLL, and Analog to Digital / Digital to Analog converters. Plugging them all together, and getting access to a real RF lab, meant that I could build the proof-of-concept and make sure that the core idea was sound.

Building a radio from discrete components turns out to be a difficult problem. The art of building a receiver is a complex and detailed one, full of tradeoffs between power, price, performance, size, and many other factors. Furthermore, a core concern for me was that I needed to build a quality transmitter as well. This ultimately turned out to be the most difficult challenge.

The first custom board, Whitebox Alpha, was built at the University of Maryland, College Park, with the help of Aaron. The build occurred four months after the first time I presented

¹System on Chip Field Programmable Gate Array



Figure 1: Whitebox Alpha

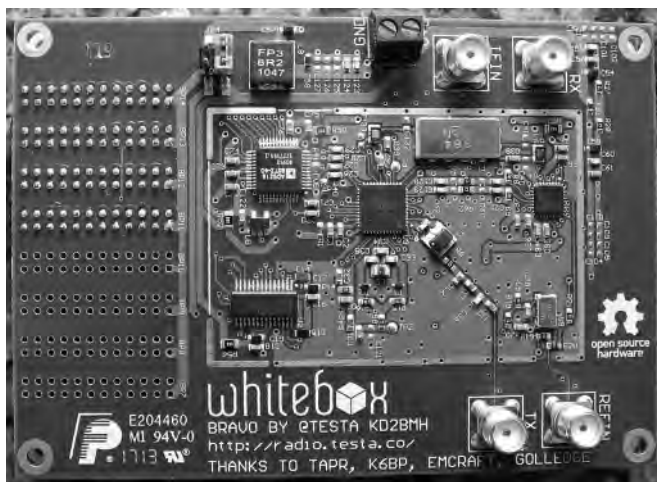


Figure 2: Whitebox Bravo

the design at DCC 2012. I fabricated two 4-layer PCBs with Sunstone Circuits, and ordered parts from major distributors in the USA including Digi-Key and Mouser. I also ordered a solder paste stencil. Most of the components were on the top, so I solder-pasted the side and then carefully placed the components with tweezers. I used a hot plate to solder on the components. It worked quite well, except for the connector for the computer, which I had to have professionally put on to get a good connection. Ultimately, the computer worked on this design but the IF oscillator was unable to lock reliably, due to me messing up the nets around the PLL Loop Filter and VCO's tank circuit.



Figure 3: Whitebox Charlie

Whitebox Bravo was built at a contract manufacturer (CM) in Carlsbad, CA, and I really enjoyed using the surface mount assembly line. This one came together around one year after the build of Whitebox Alpha. My goal here was to really focus on the core RF system and verify that I could design a PCB that would radiate RF. The design had around 130 components on it, and I was able to get all of the PLLs to lock. I began to work on the full RF signal chain. A major problem that I had still at this point was that I had no RF test gear. In particular, if you plan to make a radio without a calibrated RF Signal Generator and RF Spectrum Analyzer, I wish you luck! Those tools are critical to understand the behavior of your creation.

After a year of working on Whitebox Bravo, Bruce Perens K6BP started to acquire Boat Anchors and ship them my way to help me be able to understand the intricacies of the second prototype. The lessons were clear - the various sub-circuits need appropriate RF filtering to connect them together, and the transmitter needed additional circuitry to calibrate it for spurious emissions requirements. Given that I could solve both of those problems, the device would be ready for serious amplification and to be used by others.

Whitebox Charlie, was designed over a 7 month time span starting directly after the DCC 2014 Sunday Seminar that I did on FPGA

SoCs[5]. This board is a full five times more complicated than the previous design. The goal this time was to get the board off of my bench and into the field. It sits inside of a 160mm x 75mm extruded aluminum case from Hammond Mfg. I use U.FL connectors on every important RF net. I can always not stuff them after I've figured out the design issues. The fabricated PCB is 6 layers and the additional two microstrip layers allow for a much more dense route while maintaining signal integrity for the critical RF traces.

2.2 Baseband Subsystem

The entire design received an overhaul based on the lessons learned from the previous prototypes.

For power input, there is transient voltage suppression, reverse polarity protection, and high voltage filtering to condition the noisy signal coming from a car battery as it is charged via its alternator. Two on-board switching regulators provide efficient digital power at 3.3 and 5 Volts for the embedded computer and its peripherals. A wide input-voltage tolerant 5 Volt Low-Dropout Regulator (LDO) provides analog power, and a 3.3V regulator stems off of this for the analog circuits on the baseband. The analog portion of the board can be turned off from the microcontroller by setting a global Enable flag low. Wakeup times from this state should be on the order of 100ms. There are other standby modes available that trade wakeup times for static power dissipation.

The System on Module contains the baseband embedded ARM Cortex-M3 and FPGA [3]. It is in a separate daughter card which plugs into the main board. The main reason to not place this component myself is to not have to deal with the 484 pin BGA package, in addition to the BGA packages for the on-board LPDDR RAM (64MBytes) and Flash (16MBytes). Raspberry Pi B+ 40-pin and 8-pin (mostly) compatible headers are available for custom expansion. You'll have to try individual boards to see if they fit, and to see if you can get the driver ported. I expect most WiFi, Bluetooth, GPS, and Au-

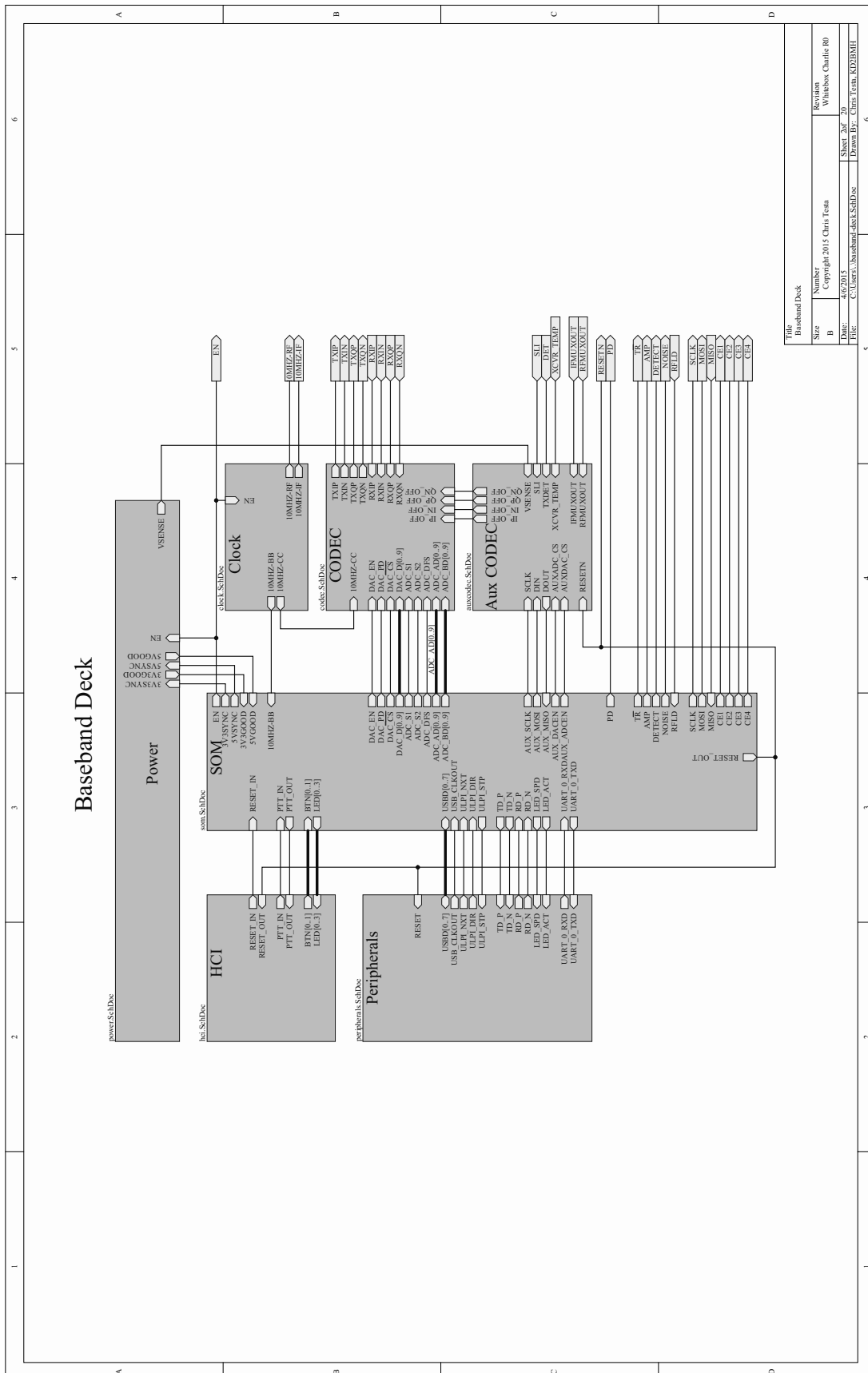


Figure 4: Baseband Subsystem Schematic

dio CODEC devices to work out of the box, but your mileage may vary. I maintain an official list of working devices in the codebase.

The system module supports standard computer peripherals including USB On The Go (OTG) and 10/100 Ethernet. There are 6 LEDs on board covering RESET, Power, PTT, and three for user control. The RESET and PTT signals also expose Open-Drain outputs so that way you can control much higher voltage equipment, like a 100W power amplifier and a T/R switch. The only externally exposed button is a RESET button, but there is a 100-mil header ready for you to tap in for Reset, PTT and dit-dah paddle inputs. All inputs are double-layer Electrostatic Discharge (ESD) protected for ruggedness.

The clocking subsystem uses a high-performance, low phase noise 10MHz Temperature Compensated Crystal Oscillator (TCXO) to provide the main sampling clock for both the analog and digital sections of the mixed signal system. A clock buffer distributes the signal to the PLL reference inputs, as well as to the ADC and DAC. A transformer coupled external 10MHz reference can be applied as well, and switched in by changing a jumper.

The CODEC is the most important set of components for transceiver performance on the baseband side of the design. For this project I am building a quadrature sampling transceiver, so the ADC and DAC must be of the dual, simultaneous sampling variety. This design features operational amplifiers at the inputs and outputs of the ADC & DAC respectively. The reason for this, which I did not understand for earlier designs, will be explained later in the section on overall system performance. A tradeoff must be made between sampling speed, sample resolution in bits, and power consumption. In this case, I chose a 10-bit ADC and am operating it at 10MSPS. The DAC is also 10-bit, 10MSPS. We would ideally move to 12-bit models, but the oversampling does help somewhat for maintaining overall system dynamic range.

An additional 8-channel auxiliary ADC is used to observe the following signals: transceiver tem-

perature, input power voltage, received signal strength, transmitter calibration signals, and PLL test points. An additional 4-channel auxiliary DAC is used to calibrate the baseband transmit signal coming out of the communication DAC. The objective of this circuit is to minimize local oscillator feedthrough. The transmitter calibration routine will be described in more detail in the next section.

2.3 RF Subsystem

The RF portion has its own power tree to isolate the subsystems as much as possible. Numerous rails are required, and LDO's were chosen for low noise and high Power Supply Rejection Ratio (PSRR). There's a 3V rail for the Low Noise Amplifier, a 3.3V Rail for the VCO's, and a separate 3.3V regulator for the rest of the analog subsystems. A 1.8V LDO supplies current to the RF Gateway.

The RF Gateway is a simple SPI slave designed in a cheap CPLD from Xilinx. The gateway talks to the main computer via SPI, and then controls the various RF and amplifier switches. This turned out to be cheaper than using discrete logic gates, and is safer than using just GPIOs. For example, it's not possible to turn on the Power Amplifier when receiving, thus reducing the likelihood of blowing the amplifiers.

Due to the superheterodyne architecture of the quadrature transceiver, two local oscillators are needed. The first oscillator works with the quadrature modulator & demodulator to go from the fixed Intermediate Frequency of 90MHz down to baseband. Since this oscillator's frequency never changes, it's Loop Filter is optimized to trade lock times for reduced phase noise.

The second oscillator works with both the receiver's mixer, as well as the transmitter's image reject up converter. This oscillator has very demanding requirements. It needs fine frequency resolution, fast lock times, and low phase noise. These conflicting requirements means that a tradeoff has to be made. Both oscillators have available U.FL connectors so a new oscillator can

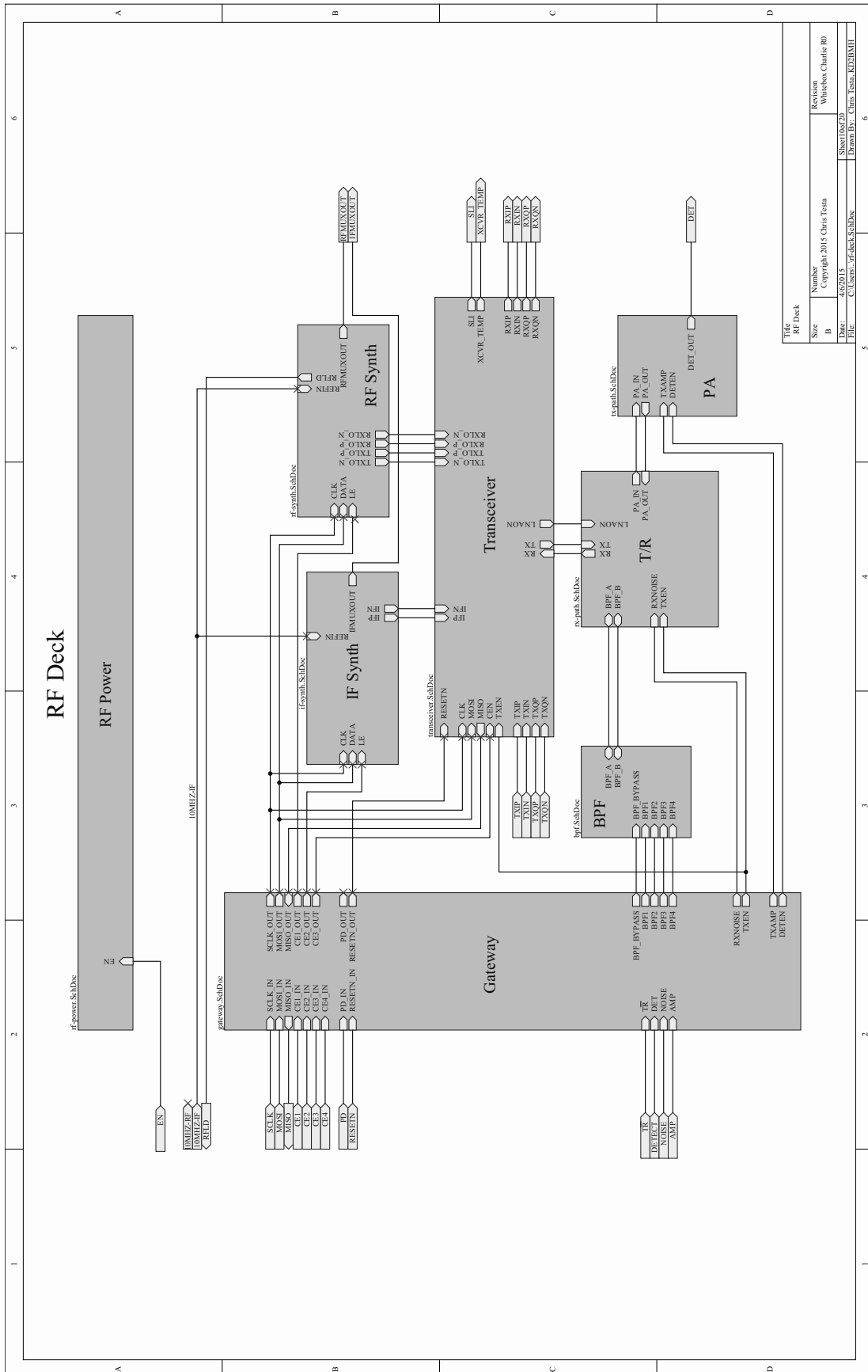


Figure 5: RF Subsystem Schematic

be plugged in depending on the application.

The core transceiver chip comes from CML Microsystems[4]. This transceiver has a very detailed manual and I've read it more times than I care to admit. There are many features of this chip and I will explore some of them later when we get to talking about the integrated design.

Between the core transceiver and the antenna jack, there sits a lot of additional RF circuitry that was not in the Bravo design. On the transmit side, after the signal leaves the transceiver chip, it flows into one of four bandpass filters in a selectable filter bank. The goal is to cut out spurious emissions that leak in from the harmonics of both oscillators.

After filtering, the to be transmitted signal can be sent down two paths: the first is to the power amplifier (20dBm maximum output) on the way to the antenna, while the second path goes to a RF log detector. The Log Detector is used to measure the signal strength of spurious emissions and is used to calibrate the transmitter's oscillator leakage and undesired-sideband suppression.

For the receiver, a switch lets you choose between two different signal sources. One comes from the transmit-receive switch, while the other comes from a built-in RF noise source. The noise source is built using an avalanche diode that is reverse biased very close to its breakdown voltage. The output of this signal is amplified and presented to the receiver chain as a self-test feature. Whichever receive signal is chosen, it then flows through the onboard bandpass filter bank and into an LNA. The LNA provides 14dB - 20dB of gain depending on the frequency. The final step as an RF signal is through a matching network on the way into the transceiver chip's mixer.

2.4 System Performance

A very important step, which was accomplished early in the Carlie design phase, was to do a full RF System analysis. The goal here is to start to look at how the transceiver would operate in a real RF link. An important thing to remember while reading this section is that when we talk

about RF signals, we really want to talk about power, and not in terms of voltage, current and resistance. So put aside Ohm's Law for the moment (though don't forget it!) and remember the power laws $P = IV = V^2/R = I^2R$. Also, we'll be using decibels everywhere, so we can just add the power terms together to get overall system power.

For the receiver, a signal is present at the antenna of lets say -110dBm at 50 Ohms. First, the signal flows through the T/R switch, the receiver signal switch, and the bandpass filter, which attenuates the signal by 6.2 dB, bringing it down to -116.2dBm. Next, the LNA is applied. The LNA provides 15 dB of gain, bringing the signal back up to -101.2dBm, while only increasing the noise by 0.6 dB.

The next sections of the receiver chain are focused around the transceiver chip. A 1:1 transformer balun and matching network brings the impedance up to 300 Ohms to be matched to the mixer. Its after this mixer stage that one of three possible bandpass filters can be selected: 1MHz, 100kHz, or 30kHz. The selected filter bandwidth plays an important role on the final Signal to Noise ratio as seen after the quadrature demodulator. This is because the narrower the IF filter, the less noise power that makes it through to the ADC.

Now comes an important step which I did not include in the Bravo design - there is an operational amplifier between the quadrature demodulator and analog to digital converter. I didn't see the purpose at first, but now I know the goal of the Op Amp is to increase the signal power. For example, if the input impedance of the OpAmp is 100kOhm, and the output impedance is 50 Ohm, and the voltage gain is set to 1x (or 0dB), there is actually a power gain of 33dB due to the impedance transformation through the voltage follower.

Overall, the receiver into the computer has a computed Noise Figure of 6dB and has sensitivity down to -110dBm, while consuming less than one Watt.

For the transmitter, the output of the Dig-

ital to Analog converter is 1 mA into a 400 Ohm resistor, or around -4dBm. There is an LC based lumped element filter followed by an OpAmp that conditions the signal leading into the transceiver chip. The image-reject up converter has a characteristic impedance of 200 Ohms, and on the way out a 4:1 balun is used to match the signal back to 50 Ohms at -10dBm. The signal attenuates 5dB as it goes through the bandpass filter bank. Next, two amplifier gain stages are applied to raise the signal up 15 and then 20 dB, resulting in a final signal strength of 20dBm, or 100mW at the antenna jack.

3 Software

3.1 User Interface

The user interface is your smartphone or tablet. My original dream was to have the smartphone interface be inside of the device, but it doesn't make sense yet to integrate it all. Step by step we can get there, but it's too complex for Charlie. Your device can be plugged into the USB OTG port and charged with the on-board 5V regulator, so they are good companions.

Android/iOS can be interfaced via USB OTG or WiFi/Bluetooth if you attach the right add-on to the Whitebox. Since I've started this project, a number of high quality Applications have been ported and implemented. AprsDroid [6] is a nice interface to APRS. Sound card support is available today, but the Bluetooth TNC or TCP modes should be possible to interface with directly given some hacking.

FLDigi [7] was ported recently and can be supported out of the box. This adds a lot of modes including MFSK, BPSK, PSK, OLIVIA, THOR, DOMINOEX, and MT63. All of these modes are supported at various standard baud rates.

There's no reason to not see the rest of the popular digital modes, like JT65 [8] ported. There's hope of getting FreeDV [9] and other digital voice modes via the Digital Voice Server [10]. I would really like to see CHIRP [11] ported to Android

with some kind of universal USB programmer. Whitebox support would be neat, too.

There's lots of fun projects for smartphones, and when we do end up with the touch screen inside of a Whitebox, all of it will work natively. So if these kinds of projects interest you, go for it!

3.2 Internal Software Stack

From the top of the software stack, the device looks like a web server over a network connection. Bruce K6BP has been contributing to the project with the Algram websocket server (checked into the main codebase[12]). He ported websockets and cJSON to the embedded platform. There's a full responsive UI for controlling the transceiver, as well as a web service API. The JavaScript supports the WebAudio API and you can control the transceiver right from Chrome on Android devices.

Available options for the receiver include a checkbox to turn on/off the LNA; 0dB - 48dB of attenuation in 6dB increments; a button to run the receiver calibration. The transmitter is supported with a checkbox to turn on/off the Power Amplifier, and a button to run the transmitter calibration. Both transmit and receive can select the appropriate bandpass filter. There are visual indicators for the transceiver temperature, input voltage, received RSSI, and PLL lock status.

The transceiver is controlled by the whitebox library. This provides the verbs and nouns needed to control the transceiver. Things like 'whitebox_tx' starts a transmission, and 'whitebox_write' writes data out to the transmitter. Conversely, 'whitebox_rx' starts a receive, and 'whitebox_read' reads data out of the receiver. There's also data structures and methods to control modems exposed from the FPGA.

Linux is the common denominator of the internal Whitebox software stack, and it provides a plethora of features. We even get the AX.25 stack for free, built right into the OS.

The kernel interfaces with all of the hardware via drivers, including a custom driver that con-

trols the digital signal processing which happens in the FPGA, which will be discussed at the end.

The driver is zero memory copy, utilizing mmap to share memory between user space and the driver. A circular buffer is used to transport data between memory to peripherals, peripherals to memory, and memory to memory with the Direct Memory Access Controller (DMA).

For connectivity, as mentioned earlier, both USB OTG and 10/100 Ethernet are available. USB OTG is probably the most interesting for expanding the Whitebox. I ported ALSA to the device, and USB sound cards work great in USB host mode. So does WiFi, Bluetooth, and GPS USB dongles.

Linux also includes a Gadget driver interface, and you can expose the Whitebox to your PC as a full USB peripheral. The same cable can support a sound card, a command line shell, a networking interface, and many more; all at the same time.

I find the Ethernet to be invaluable while I develop. You can mount your laptop over NFS to do quick and iterative development. You can also flash the operating system over Ethernet. The FPGA & bootloader can be programmed from the Ethernet too, though I have not finalized the utility for this yet.

A bricked device can be recovered via a JTAG header, though you do need a custom programmer for now. BusPirate support is coming, but it depends on Actel targeting the SVF file format for the SmartFusion2... they say it is "Coming Soon".

The FPGA toolchain is free, if you sign up with Microsemi. It supports a big enough FPGA to have a few transceivers in one. It (apparently) works on RHEL Linux, though I usually use it on Windows. You won't have to mess with that stuff though unless you want to play with the digital signal processing chain.

4 Firmware

Since the Whitebox is a quadrature transceiver SDR, an important process that happens in the baseband modem is to convert from software data, like audio or binary payloads, to a baseband signal. The baseband signal is a quadrature signal, meant to be sent through a quadrature modulator or captured from a quadrature demodulator.

To transform between the software and baseband signals, we need a modem. This modem sits in the FPGA and consists of two main parts: the digital signal converter, and the modulators/demodulators. All of them are digital signal processing flowgraphs.

The digital signal converter moves from a low sample rate baseband signal, to a high sample rate baseband signal. The signal is rate-converted with a CIC filter, and then passes through a quadrature mixer for fine tuning. The quadrature mixer is based on complex multiplies instead of CORDIC, since hardware multipliers are plentiful on the SmartFusion2. A final FIR rate converter shapes the signal up to 10MSPS. The FIR's coefficients are software controllable. The reverse flow happens on a receive.

The modem consists of both a modulator for the transmitter, and a demodulator for the receiver. I've sketched out a modem for AM, FM, SSB, and FSK, but I have not finalized the design. The full modem will sit in the smallest Microsemi FGPA with the digital signal converter by intelligently sharing the hardwired multiplier resources.

I gave the Sunday Seminar at the TAPR DCC 2014 on the concept of System on a Chip Field Programmable Gate Arrays. If you want to cover the basics, I recommend you check out the four hours of footage up on YouTube.

Since the FPGA is firmware, and it can be re-programmed in the field, it's important to have the right tools to help build the machinery in the FPGA. I have spent a lot of energy on using completely free and open tools to do all of the design validation.

The flow previously has been to use Python to describe the register transfer logic using a subset of the language and the MyHDL library[13]. This has turned out to be really valuable, as it makes generating complex Verilog modules much more streamlined. You can use object oriented constructs in Python to help efficiently describe the design.

The easiest way to do simulations is to co-simulate between Python and an Open Source verilog simulator, like Icarus Verilog[14]. This works pretty well, but it is not efficient at all. As the modem gets more complex, the simulation times grow, and it gets harder and harder to properly design the modem.

I am now using an additional tool - Verilator[15]. Verilator takes the final Verilog code, and converts it into a C++ class. Operating at the bare metal has given me a full 100x improvement in speed. Its not an Apples to Apples comparison, but at the end of the day using the new tool flow you can much more quickly and iteratively design new signal processing flowgraphs in the FPGA.

5 Conclusion

The problem of building a completely self-contained, portable software defined radio has been explored. The evolution of the hardware was documented over the three years of development. The details of the hardware for the most recent prototype were presented. The user interface and developer software stacks were covered. Finally, the digital signal processing firmware optimizations were discussed.

There are many sub-problems to explore as the hardware, software, and firmware continue to evolve and mature into a state that we all can use in the field. If you're interested in helping out in any way, contact me, visit my website[16], and get involved!

References

- [1] Testa, Chris KD2BMH. "Design of a Practical Handheld Software Radio." Digital Communications Conference 31 (2012): 122-7. Print.
- [2] Blossom, Eric. "Exploring GNU Radio". Web. 17 Aug. 2015.
- [3] Microsemi SmartFusion System-on-Module (SOM). Web. 17 Aug. 2015. <http://www.emcraft.com/products/133>
- [4] "CMX991 - RF Quadrature Transceiver." CML Micro Systems. Web. 17 Aug. 2015.
- [5] Testa, Chris KD2BMH. "System on a Chip - FPGA Programming for Mixed Signal Systems." HamRadioNow, 5 Jan. 2015. Web. 17 Aug. 2015. http://arvideonews.com/hrn/HRN_Episode_0185.html.
- [6] Lucas, Georg DO1GL. "APRSdroid - APRS for Android." Web. 17 Aug. 2015. <https://aprsdroid.org/>.
- [7] Douyere, John VK2ETA. "AndFlmsg - Flmsg with Fldigi Modems on Android - User's Manual V Beta-0.4.0." Index of /vk2eta. 21 Feb. 2015. Web. 17 Aug. 2015. <http://www.w1hkj.com/vk2eta/>.
- [8] "JT65 HF JT65A HF Frequencies Frequency Information - Digital Mode Software Download." JT65 HF. HFpack Inc. Web. 17 Aug. 2015. <http://hfink.com/jt65/>.
- [9] "FreeDV: Digital Voice for HF." FreeDV. Web. 17 Aug. 2015. <http://freedv.org/>.
- [10] Perens, Bruce K6BP. "Algoram Digital Voice Server." Algoram Digital Voice Server. Web. 17 Aug. 2015.
- [11] Smith, Dan. "CHIRP." Home. Web. 17 Aug. 2015. <http://chirp.danplanet.com/>.

- [12] Testa, Chris KD2BMH. "testaco/whitebox." Github.com. Web. 17 Aug. 2015. <https://github.com/testaco/whitebox>
- [13] "MyHDL from Python to Silicon!" MyHDL. Web. 17 Aug. 2015. <http://www.myhdl.org/>.
- [14] "Icarus Verilog." Icarus Verilog Homepage. Web. 17 Aug. 2015. <http://iverilog.icarus.com/>
- [15] "Intro - Verilator." Veripool. Web. 17 Aug. 2015. <http://www.veripool.org/wiki/verilator>
- [16] "Whitebox Bravo documentation." Testa.co. Web. 17 Aug. 2015. <http://radio.testa.co/>.

Software Defined Radio Server

“A Radio Server for VHF+ Contesting
And Weak Signal Work”

Phil Theis K3TUF

Digital Communications Conference
October 10, 2015

Initial Plans

- Need Band Data
- Switch Transverters
- 6700 is Great Radio (#1 on Sherwood Engineering List)
- No way to change uW bands
- Of HF bands for that matter

Put an Embedded Device to work

- Select Device
- Use Rapid Development Tools
 - Python
- Get on the air
- End of Story ?

Python in Action



Elegance and Simplicity

- Integrated Development Environment
- Built In – Off the Shelf
 - Beagle Bone Black
 - Immediate Bone Script
 - Python
 - Ethernet or USB

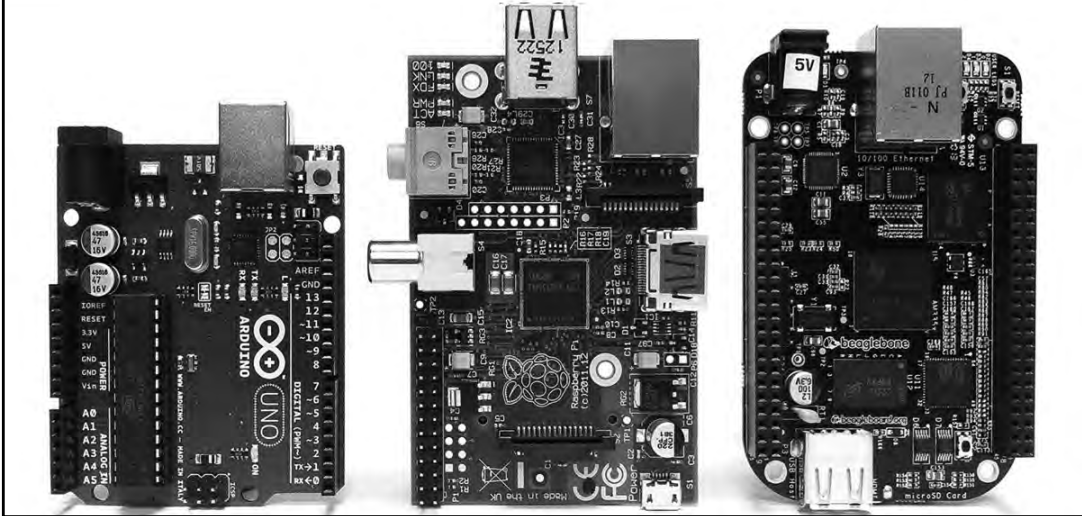
October 2014

Talk Today

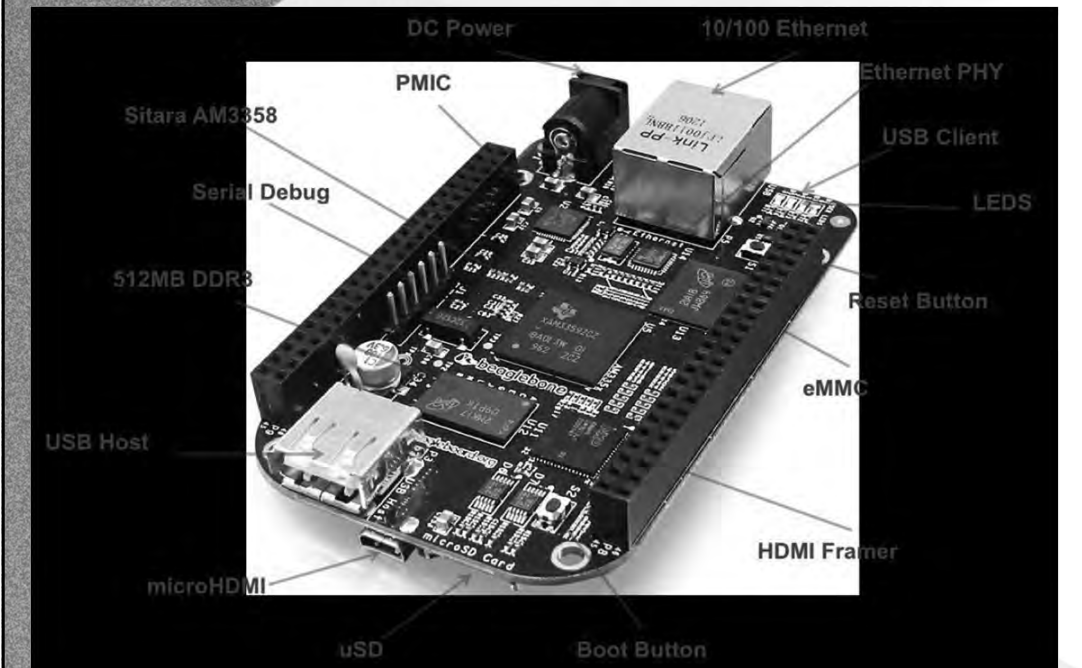
- Take you through the Process
- See what I learned along the way
- Much more that can happen
 - Transverter Control
 - Remote Control of 6K radios
 - Tasks around the Shack
- All Via Ethernet

Device Choices

- Arduino – Raspberry PI – Beagle Bone



Beagle Bone Black

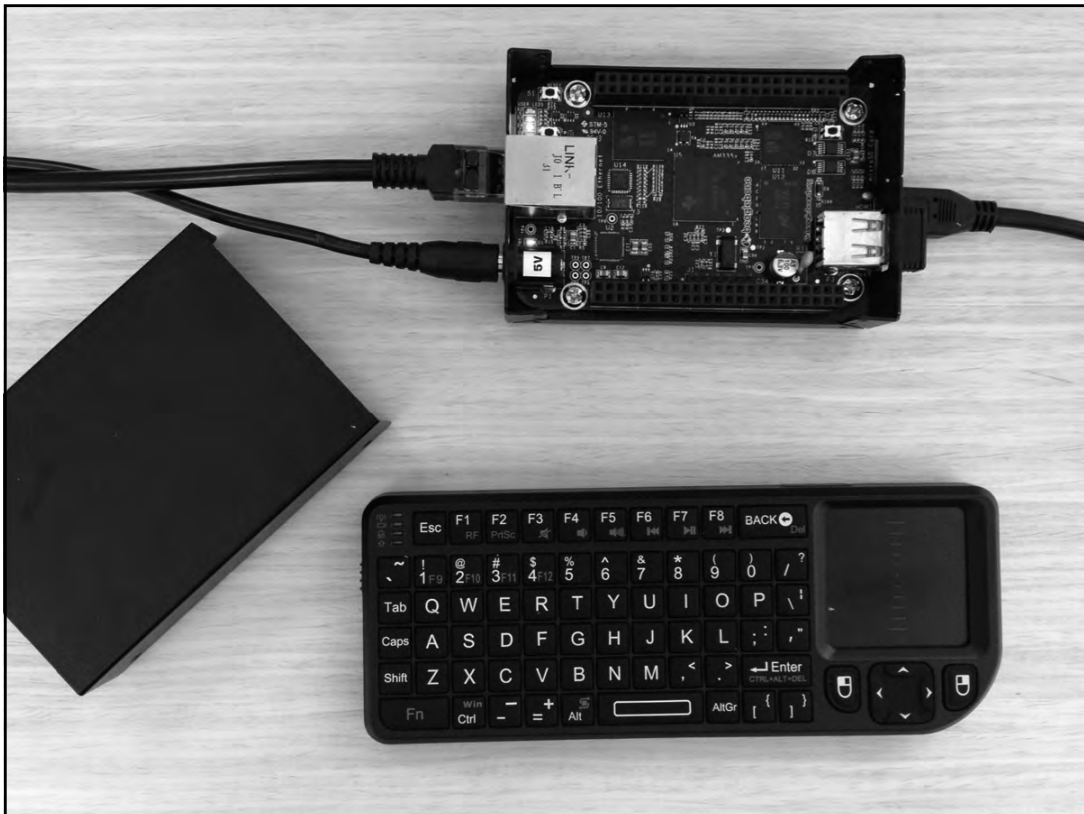


GPIO pins

65 possible digital I/Os

| P9 | | | | P8 | | | |
|------------|----|----|------------|---------|----|----|---------|
| DGND | 1 | 2 | DGND | DGND | 1 | 2 | DGND |
| VDD_3V3 | 3 | 4 | VDD_3V3 | GPIO_38 | 3 | 4 | GPIO_39 |
| VDD_5V | 5 | 6 | VDD_5V | GPIO_34 | 5 | 6 | GPIO_35 |
| SYS_5V | 7 | 8 | SYS_5V | GPIO_66 | 7 | 8 | GPIO_67 |
| PWR_BUTTON | 9 | 10 | SYS_RESETN | GPIO_69 | 9 | 10 | GPIO_68 |
| GPIO_30 | 11 | 12 | GPIO_60 | GPIO_45 | 11 | 12 | GPIO_44 |
| GPIO_31 | 13 | 14 | GPIO_40 | GPIO_23 | 13 | 14 | GPIO_26 |
| GPIO_48 | 15 | 16 | GPIO_51 | GPIO_47 | 15 | 16 | GPIO_46 |
| GPIO_4 | 17 | 18 | GPIO_5 | GPIO_27 | 17 | 18 | GPIO_65 |
| I2C2_SCL | 19 | 20 | I2C2_SDA | GPIO_22 | 19 | 20 | GPIO_63 |
| GPIO_3 | 21 | 22 | GPIO_2 | GPIO_62 | 21 | 22 | GPIO_37 |
| GPIO_49 | 23 | 24 | GPIO_15 | GPIO_36 | 23 | 24 | GPIO_33 |
| GPIO_117 | 25 | 26 | GPIO_14 | GPIO_32 | 25 | 26 | GPIO_61 |
| GPIO_125 | 27 | 28 | GPIO_123 | GPIO_86 | 27 | 28 | GPIO_88 |
| GPIO_121 | 29 | 30 | GPIO_122 | GPIO_87 | 29 | 30 | GPIO_89 |
| GPIO_120 | 31 | 32 | VDD_ADC | GPIO_10 | 31 | 32 | GPIO_11 |
| AIN4 | 33 | 34 | GNDA_ADC | GPIO_9 | 33 | 34 | GPIO_81 |
| AIN6 | 35 | 36 | AIN5 | GPIO_8 | 35 | 36 | GPIO_80 |
| AIN2 | 37 | 38 | AIN3 | GPIO_78 | 37 | 38 | GPIO_79 |
| AIN0 | 39 | 40 | AIN1 | GPIO_76 | 39 | 40 | GPIO_77 |
| GPIO_20 | 41 | 42 | GPIO_7 | GPIO_74 | 41 | 42 | GPIO_75 |
| DGND | 43 | 44 | DGND | GPIO_72 | 43 | 44 | GPIO_73 |
| DGND | 45 | 46 | DGND | GPIO_70 | 45 | 46 | GPIO_71 |

In GPIO mode, each digital I/O can produce interrupts.



Apache Web Server



- Port 80
- PHP
- Available to any Device



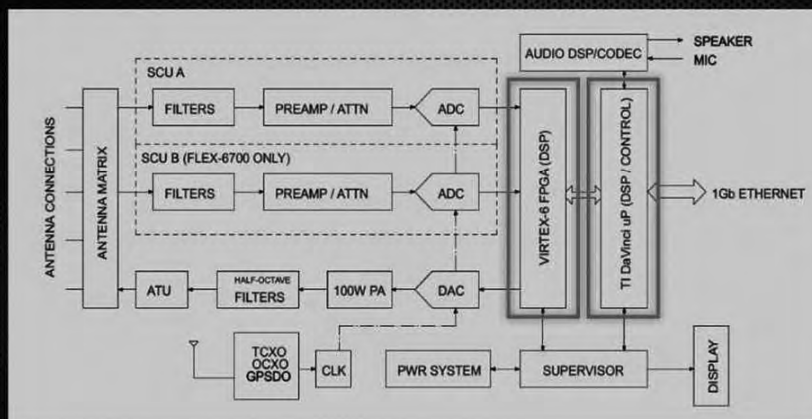
The Radio Server



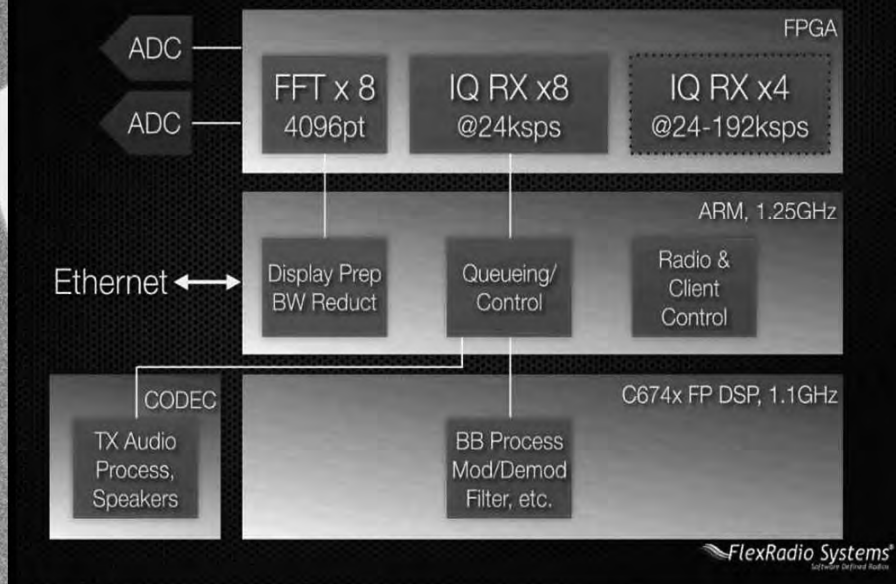
FLEX-6000 Signature Series Family

| | Pan /RX | Max BW | DAXIQ | SCU | ATU | GPSDO | Mic | Freq |
|------------|---------|--------|----------------|-----|-----|-------|-------------|--------|
| FLEX-6300 | 2 | 7 MHz | 2 up to 96kHz | 1 | Opt | — | Unbal | 160—6m |
| FLEX-6500 | 4 | 14 MHz | 4 up to 192kHz | 1 | YES | Opt | Unbal + Bal | 160—4m |
| FLEX-6700 | 8 | 14 MHz | 4 up to 192kHz | 2 | YES | Opt | Unbal + Bal | 160—2m |
| FLEX-6700R | 8 | 14 MHz | 4 up to 192kHz | 2 | N/A | Opt | N/A | 160—2m |

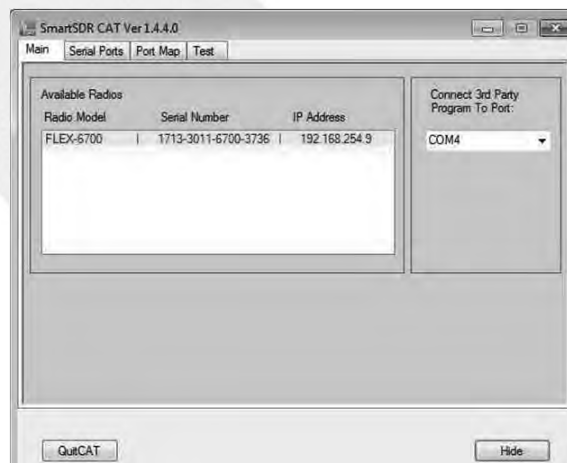
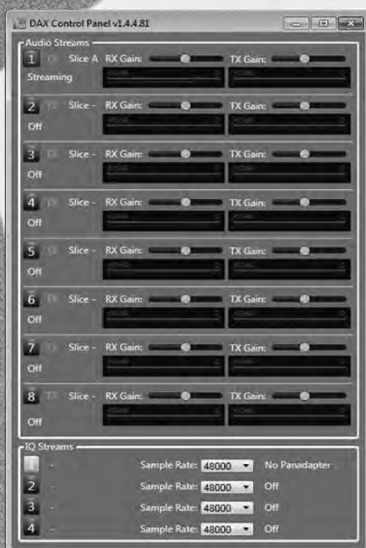
FLEX-6000 HW System Architecture



SmartSDR SW Architecture

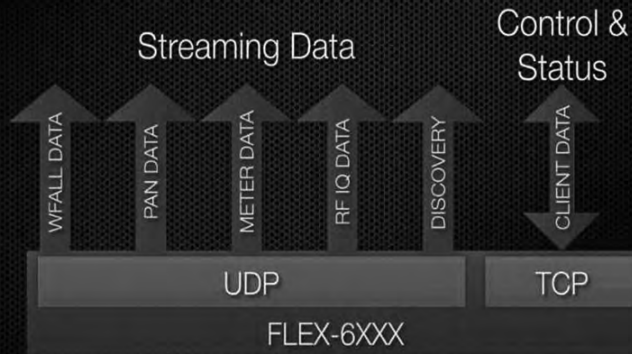


DAX & SmartCAT



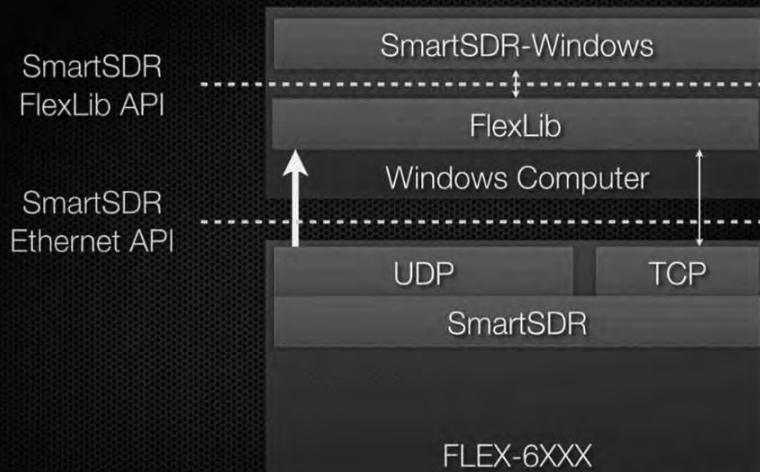
Talking to the Radio Server

SmartSDR Ethernet API Interfaces



SmartSDR and the use of FlexLib

SmartSDR APIs



Flex Uses the API

- SmartSDR Windows client rests on FlexLib which rests on the internet API
- CAT and DAX also use FlexLib
- You can do anything done in SmartSDR
- Unprecedented control over a Radio Server

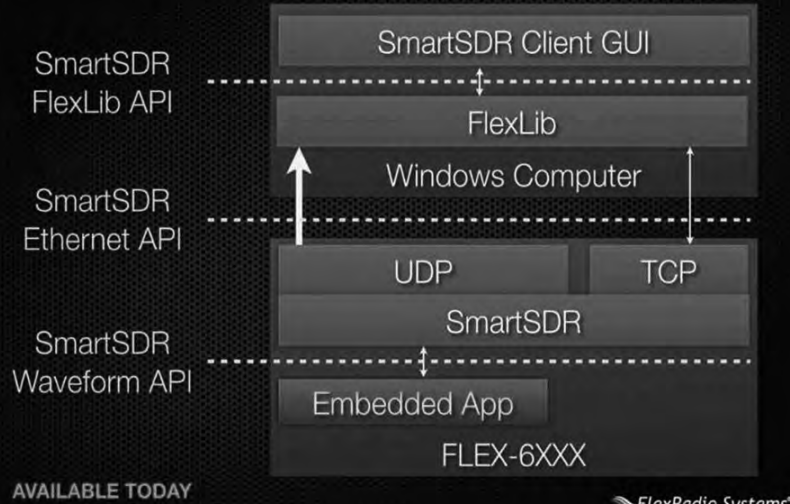
FlexLib

FlexLib - SmartSDR in .NET

- ▶ FlexLib is a .NET 4.0 DLL that provides .NET style access to the SmartSDR Internet API
- ▶ Simplifies interoperation with the radio in .NET environment - Object Oriented, Events, etc.
- ▶ Provided at no charge on the FlexRadio website

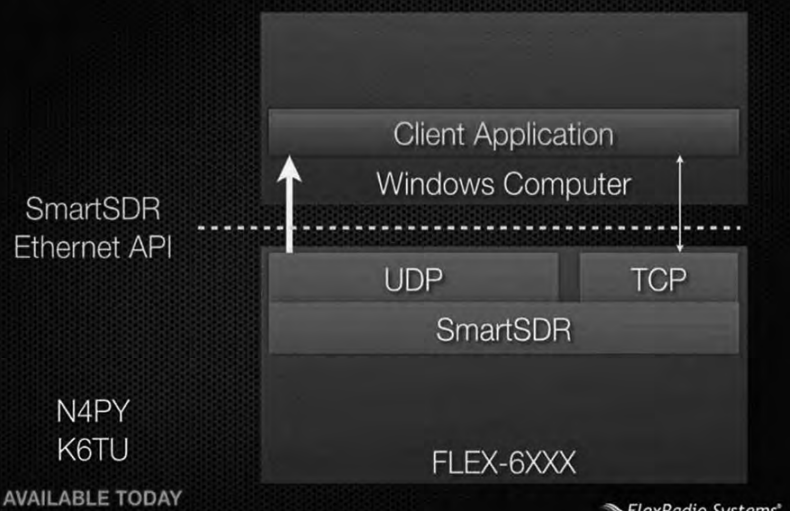
Installing App in Radio

3rd Party Embedded App



What I am doing


3rd Party App using Ethernet API



API Objectives

SmartSDR API Objectives

- ▶ Provide a common interface for FlexRadio products
- ▶ Support the building of an ecosystem around SmartSDR for the benefit of customers, developers and FlexRadio
- ▶ Provide a way to use a FLEX-6000 in a variety of applications, even ones that may not be mainstream

 FlexRadio Systems[®]
Software Defined Radio

How to talk to the API

API Standards

- ▶ Radio control is a TCP/IP socket with simple commands (no standard known):

```
slice create freq=14.1 ant=ANT1 mode=USB  
slice tune 0 14.105
```
- ▶ Streaming Panadapter/Waterfall/Meter/Discovery data are VITA-49 Extension
- ▶ I/Q and Real IF is VITA-49 IF Data (24-192ksps)

 FlexRadio Systems[®]
Software Defined Radio

API Commands

SmartSDR TCP/UDP API Command Format

- ▶ Command preface, sequence, v-bar, command
`C134|slice create freq=7.243`
- ▶ Response preface, sequence, v-bar, response
`R134|50000002`
- ▶ Status preface, handle, v-bar, status
`S67EF9A22|slice 0 freq=7.243`
`S67EF9A22|slice 0 filter_lo=300 filter_hi=2700`

 FlexRadio Systems
Software Defined Radio

Establishing Connection

SmartSDR TCP/UDP API Connecting to radio

- ▶ TCP/IP socket connection to port 4992
- ▶ API provides API version and a “handle”
`V1.1.0.0`
`H35E61405`
- ▶ Send commands!
- ▶ Interface is asynchronous, commands are non-blocking

 FlexRadio Systems
Software Defined Radio

Slice Exchange

Slice Receivers, example

- ▶ Create a slice receiver

```
slice create [freq=<MHz>] [ant=<antenna>] [mode=<mode>]  
C34|slice create freq=14.236 mode=FDV  
R34|0
```

- ▶ Tune a slice receiver

```
slice tune 0 [freq=<MHz>] [ant=<antenna>] [mode=<mode>]  
C45|slice 0 freq=14.236
```

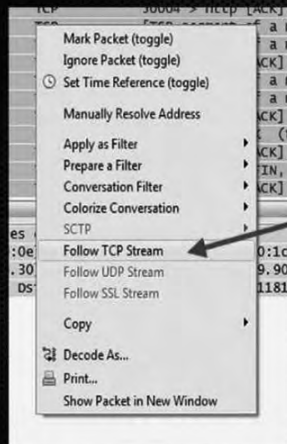
- ▶ Change slice receiver settings

```
slice set <slice> [<feature>=<value>]  
C71|slice set 0 diversity=1 tx=0 record=1  
R71|0
```

FlexRadio Systems®
Software Defined Radio

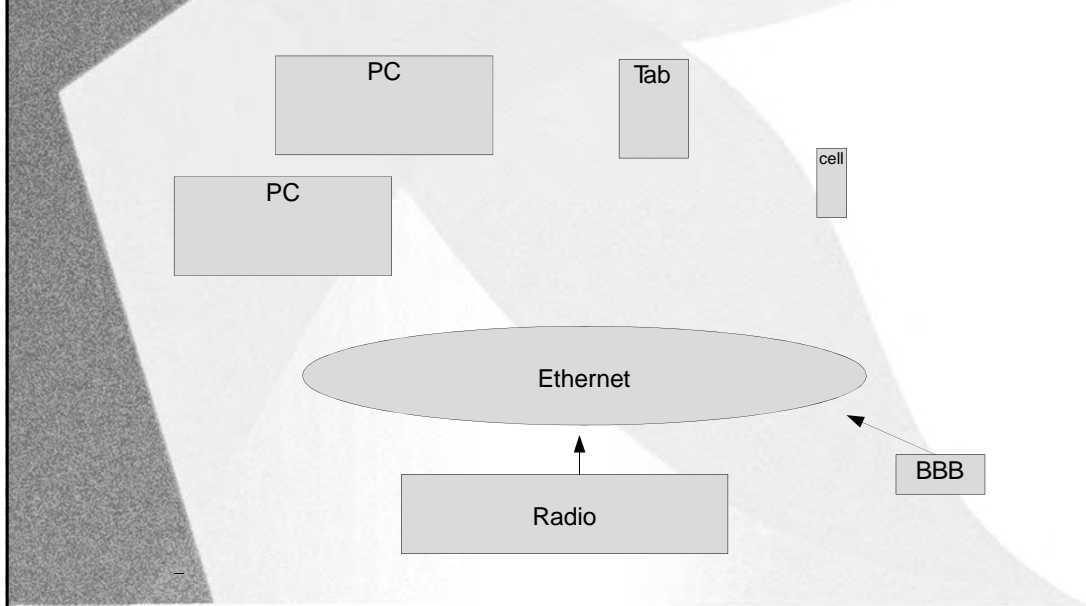
Learning the Protocol

Sniffing TCP/IP API Using Wireshark



FlexRadio Systems®
Software Defined Radio

My Port 80 Plan



Technology: Languages

- HTML Hyper Text Markup Language
- AJAX Asynchronous JavaScript and XML
- DOM The Document Object Model is a platform and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents
- Apache / PHP is a server-side scripting language designed for web development but also used as a general-purpose programming language

Technology: Languages

- C Programming Language for the server
- JavaScript is a dynamic computer programming language. It is most commonly used as part of Web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed
- JSON JavaScript Object Notation
- Python for early proof of concept

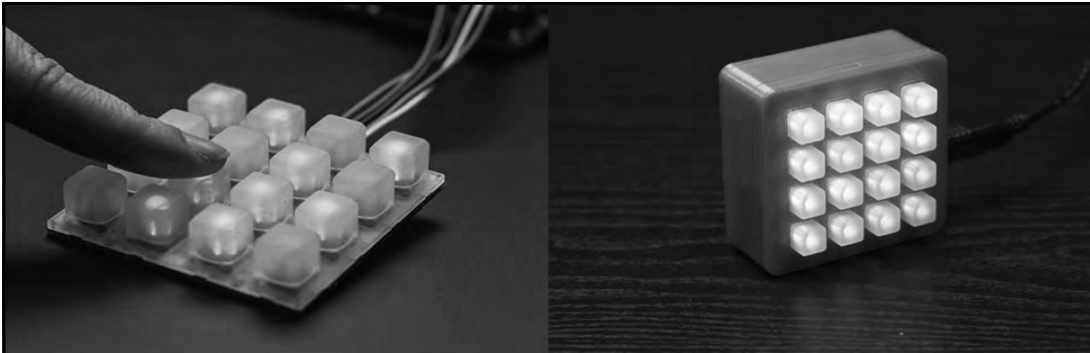
Eclipse Development Environment

The screenshot displays the Eclipse IDE interface. The main editor window shows a C program with a switch statement. The code is as follows:

```
194     else printf("%s\n", "we now have 3456");
195     currentband = 3456;
196     sendband('6');
197     break;
198
199     case 5768 :
200     if (currentband == 5768) break;
201     else printf("%s\n", "we now have 5768");
202     currentband = 5768;
203     sendband('8');
204     break;
205
206     case 10368 :
207     if (currentband == 10368) break;
208     else printf("%s\n", "we now have 10368");
209     currentband = 10368;
210     sendband('a');
211     break;
212
213     case 24192 :
214     if (currentband == 24192) break;
215     else printf("%s\n", "we now have 24192");
216     currentband = 24192;
217     sendband('b');
218     break;
219
220     case 47088 :
221     if (currentband == 47088) break;
222     else printf("%s\n", "we now have 47088");
223     currentband = 47088;
224     sendband('c');
225     break;
226
227     default :
228     if (currentband == 28) break; // 28 represents all of HP for now
229     else printf("%s\n", "must be HP");
230     currentband = 28;
231     sendband('9'); //No kramavsecc
232
233 }
234
```

The terminal window at the bottom shows the following output:

```
Last login: Mon Apr 13 02:22:10 2015 from 192.168.254.18
root@e1488-1:~# cd /usr/src
root@e1488-1:~/usr/src# ls
0970.h  bndchg.c  file1  filetest1.c  gpio.c  gpiodef1.c  makeLED.c  patternmatchpointers  patternmatchtest.c  sockettest2.c  sockettest4  strtoctest
bndchg  devmem2.c  file2  filetest2.c  gpio.h  i2c  makeLEDs  patternmatchpointers.c  sockettest  sockettest3  sockettest4.c  strtoctest.c
bndchg1.c  devmem2.c  filetest1  filetest2.c  gpiodef1.c  makeLED  makeLEDs.c  patternmatchtest  sockettest2  sockettest3.c  sockettest5.c
root@e1488-1:~/usr/src#
```



- Instantaneous Re-Configuration
- Liaison to Run
- Split Audio
- No Loss of Focus
- Complete Control of Radio
- LED Feedback

Future Tasks

- Monitor Temperatures
- Control Power Supplies
- Turn Antennas / Switch Antennas
- Multiple Locations with Distributed Computing
- Beacon Monitoring: Propagation Notification
- Performance of Beacons: Real Time Status
- Dayton Demonstration

SatNOGS: Satellite Networked Open Ground Station

Daniel J. White, Ph.D., AD0CQ
Valparaiso University
Valparaiso, Indiana
dan.white@valpo.edu

Ioannis Giannelos, Agisilaos Zissimatos, Eleytherios Kosmas,
Dimitrios Papadeas, Pierros Papadeas, Matthaios Papamathaiou,
Nikolaos Roussos, Vasileios Tsiligiannis, Ioannis Charitopoulos
Libre Space Foundation
Athens, Greece
info@satnogs.org

Abstract—The SatNOGS, or Satellite Network Open Ground Stations, project promotes and supports free and open space applications. It seeks to solve the problem of connecting many satellite users/observers to many ground station operators. Modern open software, web, and hardware techniques are used in implementing the Network, Database, Client, and Ground Station sub-projects. Modularity in all the systems promotes the dual-use of ground stations by not interfering with local operation while utilizing the great amount of time a civilian, non-commercial ground station would otherwise sit idle.

Index Terms—SatNOGS, CubeSat, software-defined radio, satellite ground station, open source

I. INTRODUCTION

The SatNOGS [1] project seeks to build a full stack of open technologies for satellite ground stations. Civilian satellite launches have been in a state of change in recent years from the introduction of very small spacecraft which use standardized launch carriers such as the CubeSat and PPOD specifications [2]. This has lowered the bar to satellite ownership and their availability for educational and amateur projects and citizen science.

Figure 1 shows that the number of CubeSat-class satellite launches has increased nearly exponentially since the first in 2000. Previously the domain

of University projects, the last 3 years have seen a huge increase in non-government or university launches. These civilian satellites include commercial, like Planet Labs' Flock-1 satellites [3], non-profit, like The Planetary Society's recent LightSail [4], and amateur, like AMSAT-NA's upcoming Fox-1 series [5].

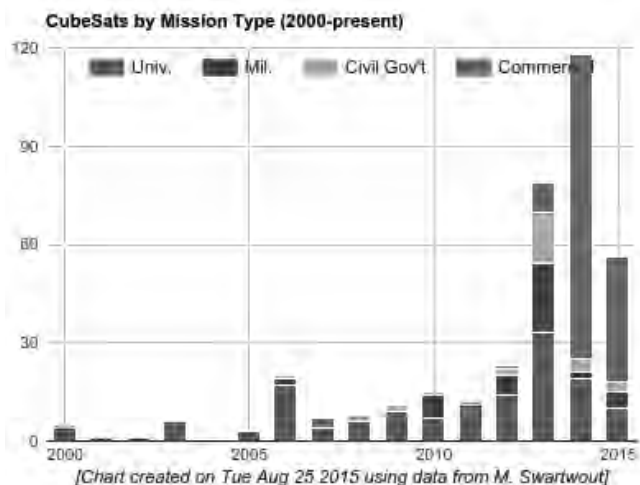


Fig. 1. CubeSat launches per year through 2015-07-17, from [6]. The "Commercial" category includes non-profit and amateur satellites.

Each satellite owner typically operates their own ground station for command and control. The low-

earth orbits (LEO) of these spacecraft result in short time windows when the spacecraft is above the local horizon for communication. As a result, owners seek to enlist the help of other suitably equipped stations for collection of data. The FUNcube project is a prime example of a well-organized effort to receive and collect data from satellite for educational outreach [7].

Recent advances in low-cost, software-defined radio (SDR) technology and 3D printing have put ground station ownership within the reach of individuals. Largely composed of Amateur Radio operators, these people receive telemetry and data from many satellites and provide the information to the owners and the general public.

Once an individual or organization builds a ground station, especially if not a commercial venture, the hardware ends up sitting idle for a great majority of the time. This capability, when not being actively used by the local owner, could be utilized for reception of other satellites. The SETI@home project is one of the earliest and well-known projects to make use of idle resources – compute cycles in that case [8].

What is missing is a civilian infrastructure to connect these many owners and ground station operators in a way that is flexible and open. The ESA’s Global Educational Network for Satellite Operations (GENSO) [9] was a notable attempt at such a network aimed at University-class projects and stations. The fact that the `genso.org` domain

name does not resolve to a live server is a practical indication of the current state of the project.

Members of *Hackerspace.gr* [10] in Athens Greece first proposed SatNOGS as a part of the 2014 International Space Apps Challenge’s “Virtual Ground Station App – Global Crowdsourcing of CubeSats” Challenge [11]. It was later submitted as an entry to the 2014 Hackaday Prize, ultimately winning the grand prize of \$196,418 [12]. The funds provided seed money to start the Libre Space Foundation [13], a new non-profit dedicated to supporting free and open source space and related projects.

A timely and unique aspect of the project is its bridging of the Maker / Hacker and the Amateur Radio communities. The 2015 TAPR/AMSAT Banquet’s speaker, Michael Ossmann AD0NR pointed out the many characteristics valued and shared by these two communities [14]. Ward Silver, N0AX’s article for Makezine also does a good job of drawing these connections [15].

This paper seeks to give an overview of the SatNOGS project’s major components. Figure 2 gives a high-level overview of the relationship between users and ground stations. Figure 3 and the following sections describe the four major sub-projects: Network, DB, Client, and Ground Station. The modular approach maximizes use of already available components at a ground station.

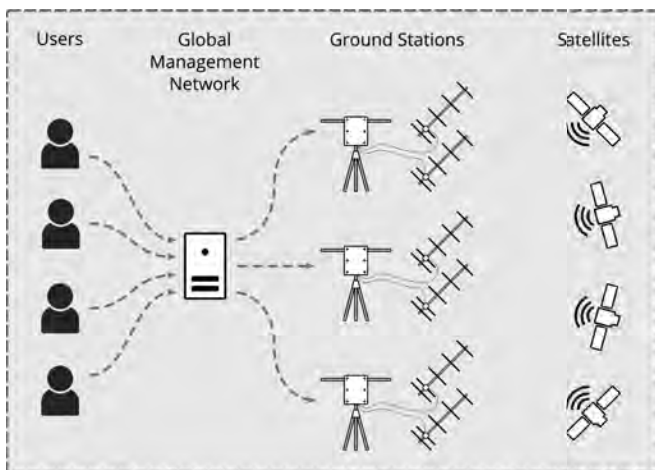


Fig. 2. Overall view of the SatNOGS concept.



Fig. 3. The four sub-projects are designed with a modular approach with well-defined interfaces, allowing the ability to relatively easily interface with existing ground stations.

All code and hardware designs for the project are publicly available under Open Software (AG-PLv3 and GPLv3) and Open Hardware (CERN OHLv1.2) licenses [16]. Software documentation may be found at [17]. Note, screen captures of the various software components, hardware pictures, and demonstrations are included in the more appropriate format of the presentation instead of in the paper.

II. NETWORK

The SatNOGS Network is accessed by users via a web interface. The user provides details about observation that they would like to schedule (which satellite, which band, time-frame, signal encoding, etc.). From this information, the system calculates the possible observation windows from the currently available Ground Stations connected to the Network having the necessary capabilities. Once the observer confirms the proposed “Observation Job” then it is sent as a job to each Ground Stations’ job queue to be executed. Figure 4 shows this process as a diagram.

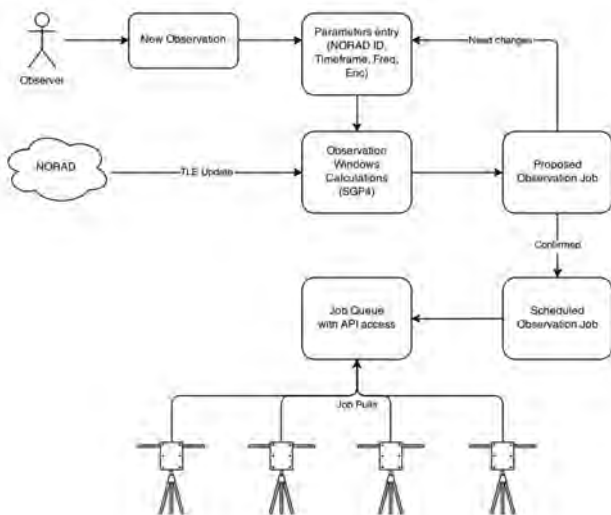


Fig. 4. Diagram of a user scheduling an observation on the network.

Ground Stations collect observation data then send it back to the Network. Uploaded data is then made available to the initiating user and any other third party via the observation’s ID. Modern web technologies are used on the Network website to provide timeline and recording visualization

and playback directly in the browser. Download links are also provided for further offline analysis. No special software or installation is necessary for users to interact with the Network. SatNOGS Network provides an API to allow other applications and services to query information and, in the future, automatically schedule observations.

Calculation of candidate times when the target satellite is visible from an active ground station is performed with the assistance of the PyEphem library [18]. The library accepts orbital elements for the satellite, Ground Station locations, and the desired time frame. With higher densities of Ground Stations which can see multiple satellites, this scheduling can be optimized for many factors.

Because all code and documentation of all parts of the project are free and publicly available, anyone is able to contribute to these improvements. Indeed, this open collaboration is one of the SatNOGS project’s founding principles.

III. DATABASE

The coordination and aggregation provided by the Network component requires a centralized source of satellite information such as frequencies and transmission modes. SatNOGS DB was created to address the fact that there was no known public source for this information. In the same spirit of the rest of the project, the database is open access and not specific only to the SatNOGS project.

Specifically, DB is a crowd-sourced suggestions app for transponder data. Satellites are identified by their NORAD (now USSPACECOM) space object catalog number and their common name. Each object has an associated set of transponder records which indicate frequencies and modulation formats. SatNOGS Network pulls this information when calculating possible observations.

Updates are accepted from the public and from other sources of satellite information with open APIs. As with the Network, user interaction is purely web-based and requires no additional software besides a capable browser.

IV. CLIENT

SatNOGS Client consists of software running a computer which controls the ground station hardware. Figure 5 diagrams the internal (modular!)

components of the client's interaction with the Network.

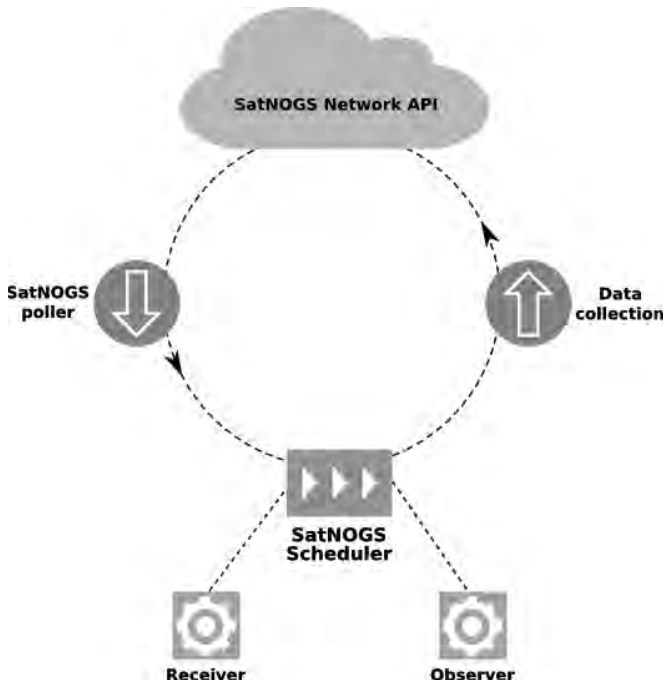


Fig. 5. Client software components and interactions.

The poller regularly checks the Network API for observation jobs scheduled for the local Ground Station. Information contained in each job includes satellite orbital elements, receiver tuning and demodulation parameters, and timing. The PyEphem [18] library is used to calculate the necessary antenna pointing and relative velocity for doppler shift calculation during the pass.

Where possible, existing standard interfaces are used within the Client. These include using the daemons `rotctl` for rotator control and `rigctl` for frequency control from HamLib [19]. Any rotator controller which works with `rotctl` automatically works with the SatNOGS Client by providing the appropriate IP address and port number the rotator is listening on.

External programs for starting the receiver driver are spawned before a scheduled observation and setup to listen for `rotctl` commands for doppler frequency corrections throughout a pass. For use with the popular Realtek RTL2832U DVB-T dongles, the SatNOGS group maintains a version of `rtl-sdr` [20] software's `rtl_fm` utility which accepts frequency control via `rigctl` commands

over a network port [21]. Code for interfacing the Client with other SDR software is in progress, including receivers using GNU Radio [22]. Radios whose baseband signals enter the computer via soundcard are also planned.

At the end of a pass, demodulated signal data is placed in a queue for later upload to the Network. Logs and other reports are also sent back to the Network by other API actions.

V. GROUND STATION

The SatNOGS Ground Station sub-project encompasses the antennas, rotator, and RF path. Ground station owners execute the Client software and build or configure the hardware. Plans and instructions are available for the v2 ground station and are being finalized for the updated v3 version. Gears and other parts are 3D printed and the rest of the hardware for the ground station is easily available. Besides access to a 3D printer, only hand tools are required for building either version.

Because the Ground Station rotator's controller implements the common EasyComm 2 protocol, it acts like any other homebrew or commercial rotator controller using the same format. The Client software therefore works exactly the same with the SatNOGS rotator design or with an existing station's rotator.

There are designs for every component necessary for a SatNOGS-compatible ground station as part of the project. Antennas and RF path hardware may be use SatNOGS, homebrew, or commercial as the station owner deems appropriate. To help the goal of easily constructed ground stations, the project's designs focus on common hardware, 3D-printed parts, and tools available in most experimenters' shops of local "maker spaces."

VI. CONCLUSION

The SatNOGS project aims to provide and promote free and open source satellite ground stations. Modern open software and web technologies are used to coordinate these stations to more fully utilize the reception capabilities for low earth orbiting satellites. By using a modular approach to the ground station segment, the existing stations of radio amateurs and others may be used with the network. Avoiding custom, network-specific

software and hardware and ensuring all design information, code, and received data is and remains freely available is a core tenet of the project. Individuals and organizations are encouraged to partner with the project to help realize these goals.

REFERENCES

- [1] SatNOGS — Satellite Networked Open Ground Station. <http://satnogs.org>
- [2] CubeSat — Wikipedia. <https://en.wikipedia.org/wiki/CubeSat>
- [3] Planet Labs. <https://www.planet.com/>
- [4] The Planetary Society. <http://planetary.org/>
- [5] AMSAT North America — The Radio Amateur Satellite Corporation. <http://www.amsat.org/>
- [6] M. Swartwout. CubeSat Database. <https://sites.google.com/a/slu.edu/swartwout/home/cubesat-database>
- [7] The FUNcube Project. <http://funcube.org.uk/>
- [8] SETI@home. U.C. Berkeley. <http://setiathome.berkeley.edu/>
- [9] European Space Agency. Global Educational Network for Satellite Operations. http://www.esa.int/Education/Global_Educational_Network_for_Satellite_Operations
- [10] Hackerspace.gr. Athens, Greece. <https://www.hackerspace.gr/>
- [11] <https://2014.spaceappschallenge.org/project/satnogs/>
- [12] Hackaday. The 2014 Hackaday Prize. <https://hackaday.io/prize/2014>
- [13] Libre Space Foundation. <http://docs.satnogs.org/>
- [14] HamRadioNow. Adventures of a Hacker Turned Ham, Michael Ossmann. TAPR/AMSAT Banquet, Dayton 2015. <https://www.youtube.com/watch?v=LpSIGKqeZ4I>
- [15] W. Silver NOAX. A Maker's Introduction to Ham Radio. Makezine. <http://makezine.com/2015/06/30/a-makers-introduction-to-ham-radio/>
- [16] SatNOGS Code and Documentation Repository — GitHub. <https://github.com/satnogs>
- [17] SatNOGS Documentation. <http://docs.satnogs.org/>
- [18] PyEphem. <http://rhodesmill.org/pyephem/>
- [19] hamlib — Ham Radio Control Libraries. <http://sourceforge.net/projects/hamlib/>
- [20] Osmocom. rtl-sdr. <http://sdr.osmocom.org/trac/wiki/rtl-sdr>
- [21] SatNOGS. rtl-sdr — rtl_fm with rigctl. <https://github.com/satnogs/rtl-sdr>
- [22] GNU Radio. <http://gnuradio.org/>

ISBN: 978-1-62595-040-6

52000



9 781625 950406

ISBN: 978-1-62595-040-6