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 FPGA[1], 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

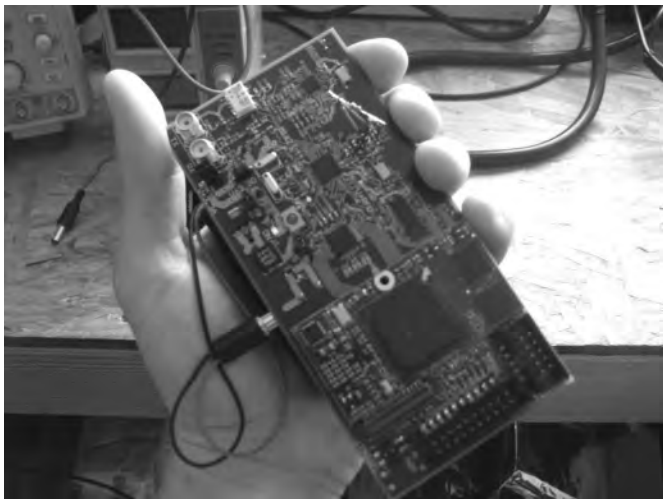
Figure 1: Whitebox Alpha

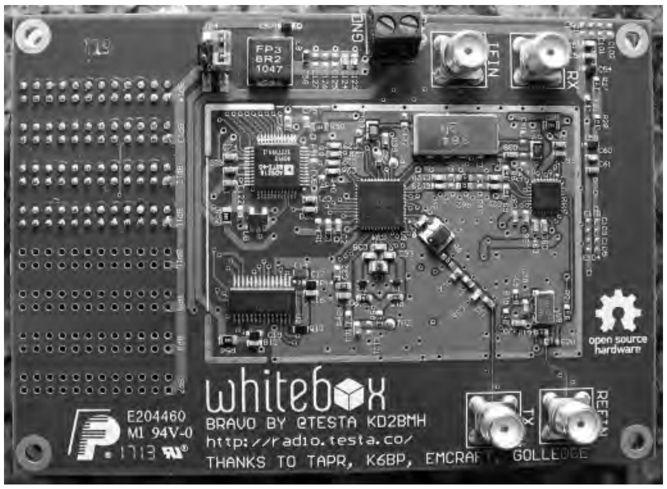

Figure 2: Whitebox Bravo

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.

[1]System on Chip Field Programmable Gate Array

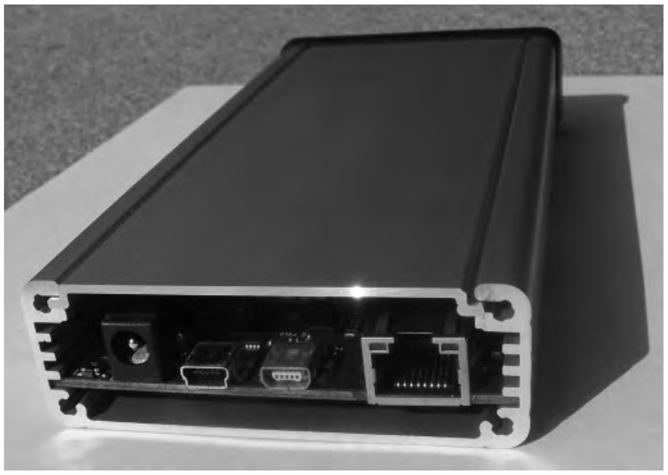
Figure 3: Whitebox Charlie

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 subcircuits 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-

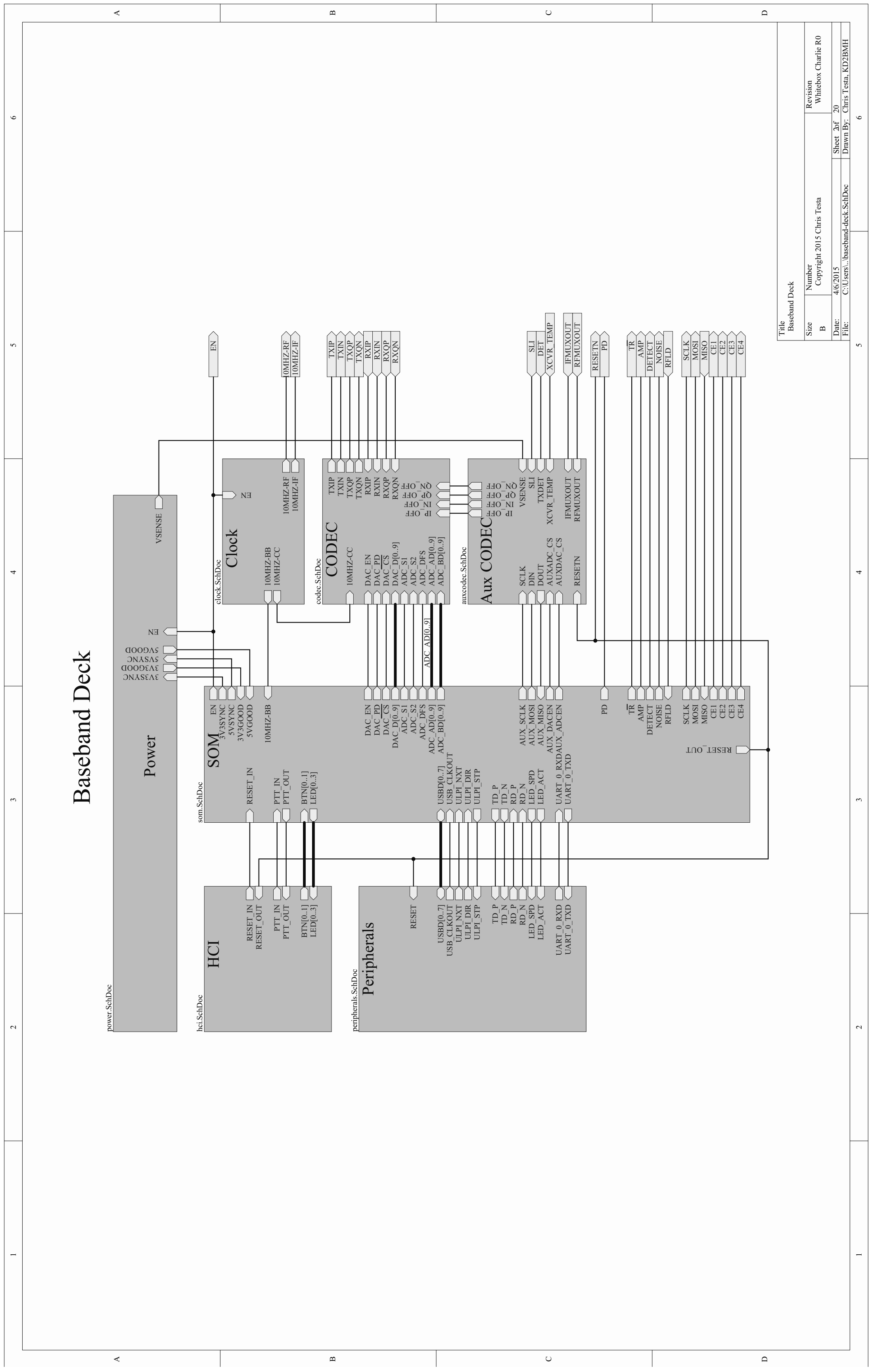
Baseband Deck
Power
SOM
Clock
CODEC
Aux CODEC
HCI
Peripherals
Title Baseband Deck
Number Copyright 2015 Chris Testa
Revision Whitebox Charlie R0
Size B
Date: 4/6/2015
Sheet 2of 20
File: C:\Users\..\baseband-deck.SchDoc
Drawn By: Chris Testa, KD2BMH

Figure 4: Baseband Subsystem Schematic

dio CODEC devices to work out of the box, but your mileage may vary. I maintain a 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 temperature, 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

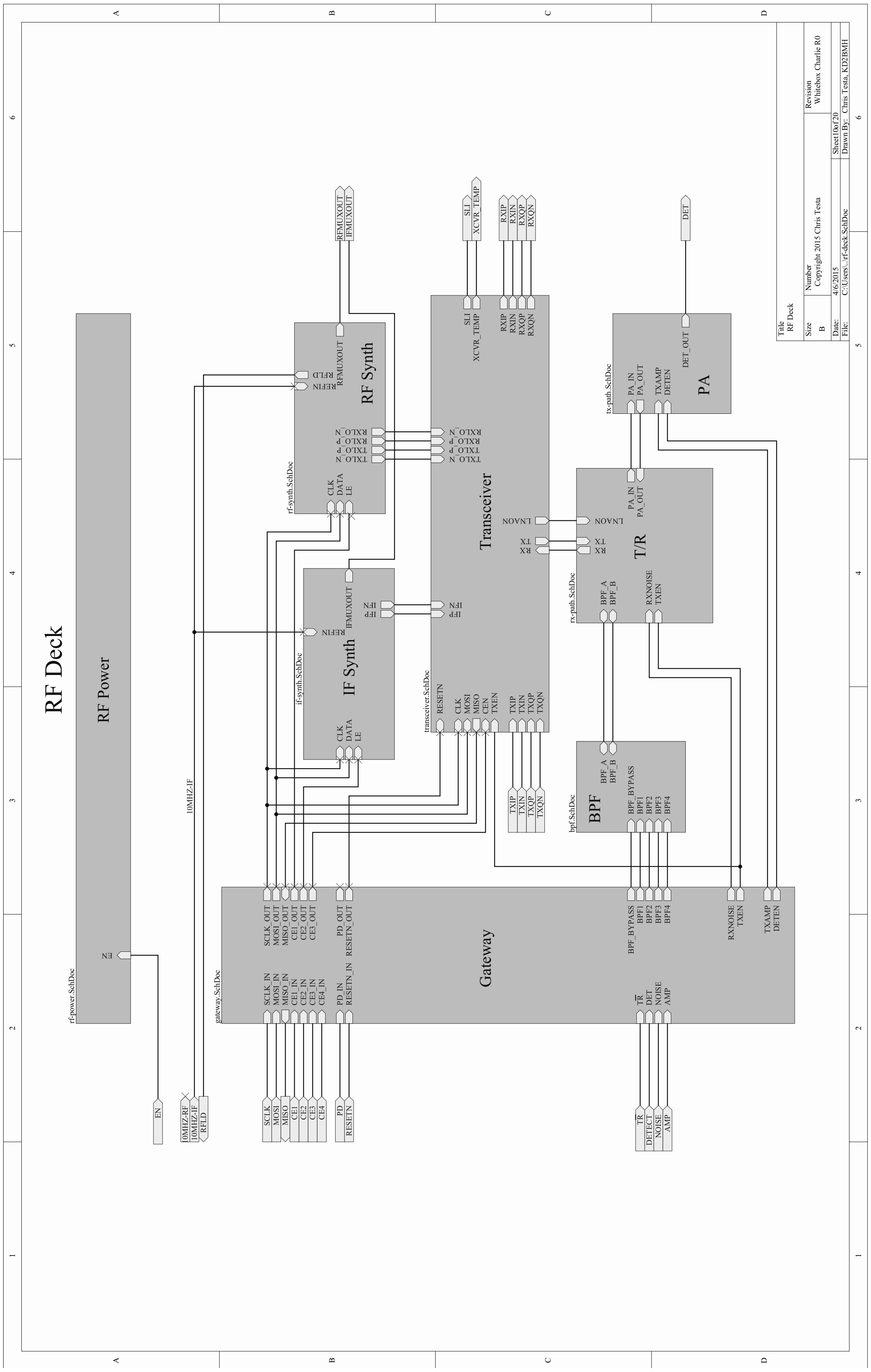
RF Deck
RF Power
RF Synth
IF Synth
Transceiver
T/R
PA
BPF
Gateway
Title RF Deck
Number Copyright 2015 Chris Testa
Revision Whitebox Charlie R0
Size B
Date: 4/6/2015
Sheet 10of 20
File: C:\Users\..\rf-deck.SchDoc
Drawn By: Chris Testa, KD2BMH

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 Digital 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 Algoram 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 controls 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 reprogrammed 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] Lucus, 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://hflink.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/.