A PORTABLE SOFTWARE RADIO USING COMMODITY HARDWARE AND OPEN-SOURCE SOFTWARE

Michael Dickens, Brian Dunn, and J. Nicholas Laneman
University of Notre Dame, Department of Electrical Engineering, Notre Dame, IN
{mdickens, bdunn2, jnl}@nd.edu ; http://radioware.nd.edu/

ABSTRACT

We summarize the design and implementation of a portable software radio prototype built primarily using commercial off-the-shelf components and open-source software. The device components include a general-purpose processor (GPP) on a small-form-factor motherboard, radio hardware, touchscreen and LCD, audio microphone and speaker, and an internal battery enabling hours of mobile operation. Significant advances over the past decade have made GPP-based software radio a viable solution in many areas, and this work demonstrates that today's processors are capable of enabling a new generation of software radio in portable form-factor devices. Our research group leverages these prototypes for several funded projects focusing on issues including interoperable public safety communications, cognitive wireless networking, and educational initiatives.

Figure 1: Highly reconfigurable portable software radio prototype implemented using open-source software and predominantly COTS hardware. The prototype provides dynamically configurable multi-channel and full-duplex communications in most frequency bands from 50 MHz to 2.9 GHz.

1. INTRODUCTION

This article describes the design and development of a portable software radio prototype that uses as much open-source hardware and software as possible, and leverages commercial off-the-shelf (COTS) components. The device is shown in Figure 1, and operates using GNU Radio software [1] for signal processing on a small-form-factor general-purpose processor (GPP)-based computer and an Ettus USRP (Universal Software Radio Peripheral) [2] for the air interface. The prototype offers the same capabilities as GNU Radio running on an Intel Core 2 Duo CPU-based computer running at 2 GHz with an Ettus USRP attached. The device can fit inside a box of dimensions 29 cm x 10.5 cm x 21 cm, weighs just under seven pounds, and has roughly two hours of runtime from a single battery charge. The bill of materials for construction of a single device using retail components comes to approximately $3,700. The prototype described here exemplifies the benefits and cost savings offered by leveraging open-source and COTS components, and to the best of our knowledge represents the first portable software radio of its kind.

For high-volume applications, hardware radios are often preferred over software-based implementations. However, as processing performance and power efficiency increase there will be more applications for which software radios are more capable and can be built more cost effectively than their hardware counterparts. Even today, there are many applications that benefit from the superior reconfigurability offered by software-based implementations, such as when device interoperability is critical, when the lifetime of a product greatly exceeds that of the devices with which it needs to communicate, and for wireless research and development. There are also an increasing number of applications for which software-based processing is a feasible alternative and the added reconfigurability is highly desirable. Given the growing application-range for software radios, and their relatively limited presence in industry, we set out to demonstrate that today's GPPs are capable of enabling portable communication devices that offer superior flexibility and advanced functionality.

In the following section we give more specific objectives, motivate the decision to use GNU Radio as the software framework for the prototype, and discuss several alternative platforms that offer similar functionality. In Section 3 we provide details on what went into the selection and design of each hardware component along with their integration into the prototype. In Section 4 we do the same for software. We conclude in Section 5 with a summary of our work on the prototype, and a discussion of some broader implications related to this work and software radio as a whole.

2. CORE PLATFORM SELECTION

At the early stages of the project, we had two primary goals. First, we wanted to create a device that could be used as a wireless research platform to quickly try out new ideas and provide a more concrete basis for what would otherwise be purely theoretical work. A key metric of success for this goal was minimizing the learning curve and development time often associated with algorithm development and experimental work in communications. Second, we set out to demonstrate that comprehensive protocol agility through software-based processing on a mobile device is not only a technology of the future, but a viable alternative today.

A generic software radio consists of the radio hardware, optional user interfaces, and – most importantly – one or more signal processors. The primary options for each signal processor include a field-programmable gate array (FPGA), a digital signal processor (DSP), and a GPP.

We based processor selection on the anticipated uses of the prototype, e.g., by researchers without specialized programming expertise. As shown in Figure 2, GPP-based software requires the least programming specialization, provides the best code-reuse, and can also be readily modified to include new or additional functionality, e.g., upgrading software from a draft to accepted wireless standard. Another argument for using a GPP is the upgrade path to newer, more capable processors via direct replacement of the processor or the motherboard on which the processor resides. A faster processor allows current code to execute faster – possibly even in real-time – and for more sophisticated algorithms to be implemented, simply by recompiling for the new hardware. We decided that it was worth the cost in terms of reduced battery life to use a GPP, in order to achieve the high level of flexibility and ease of use that we required.

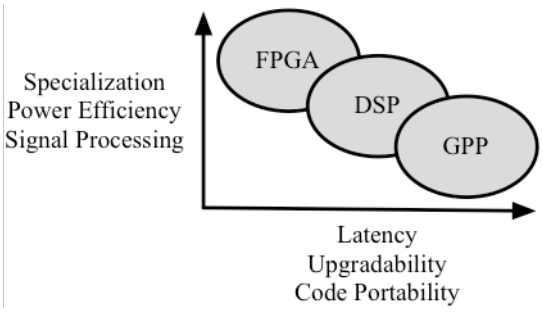

Figure 2: Graphical depiction of the tradeoff between various properties of signal processors.

Proprietary hardware and software has traditionally been required when building a software radio in order to overcome some fundamental limitations including radio frequency (RF) access range, digital data-transport bandwidth, and signal processing capabilities. Commercially-available advances in the myriad radio hardware technologies – antennas and RF front ends, analog-to-digital and digital-to-analog converters, data-transport protocols and hardware, signal processors and small-form-factor computers, and power-management systems and batteries – and the maturity of freely-available open-source radio software have significantly mitigated these limitations. Accordingly, we worked to leverage as much open-source software and COTS hardware as possible. The use of a GPP and open-source software for signal processing are key to achieving both goals by controlling hardware costs, creating a highly scalable processor upgrade path, and fostering a collaborative environment for software development in the wireless community.

We identified several baseline requirements to show that a GPP-based software radio could be built in a portable form factor offering reasonable runtime while powered from an internal battery. In order to meet the needs of a funding agency, it was important that the system be capable of handling multiple 25 kHz voice channels with typical voice encoding, and data transmission up to a few hundred kilobits per second with QAM, PSK, or OFDM modulation. With respect to software architecture requirements, we desired cross-platform support for Unix-like operating systems including Linux and Mac OS X, critical code written primarily in a compiled, standardized, operating-system independent programming language such as C++, and software-based control down to the physical layer. Finally, we wanted the hardware to be economically priced, while still offering processing performance on par with currently available desktop computers.

It was initially unclear whether we could leverage an existing software radio platform, or if it would be necessary to develop a new platform specific to our needs. To build from an established platform, we looked for mature open-source projects, focusing on software-based signal processing independent of any specific operating system. There were a variety of projects that offered some of the features we needed. Those best aligned with our goals included:

- GNU Radio software and Ettus USRP hardware;
- CalRadio [3];
- High Performance SDR [4];
- KU Agile Radio [5];
- Rice WARP [6];
- Lyrtech Small-Form-Factor SDR [7];
- University of Texas HYDRA [8];
- Virginia Tech Chameleonic Radio [9];
- Virginia Tech Cognitive Engine [10]; and
- Virginia Tech OSSIE [11].

Many of the platforms did not meet our needs for several reasons. Some were too expensive or overly bandwidth-constrained, limiting their usefulness to voice and audio transport. Others relied exclusively on FPGA-based signal processing or did not provide sufficiently open-source software. We found the combination of the GNU Radio software and Ettus USRP hardware to be the best candidate, and considered the platform to be sufficiently mature to use in our devices. In the following sections we discuss some trade-offs involved in selecting this platform and describe the hardware and software development specific to the prototype. Figure 3 shows a conceptual drawing of the complete software radio platform and each functional component. The discussion is structured such that readers who may have different requirements, constraints, or other considerations may readily map our decision processes and lessons learned to their environments.

3. HARDWARE INTEGRATION

Even with our decision to use the Ettus USRP for the radio hardware, there were still a number of issues we had to address including creation of the enclosure and audio interface, and selection of the host computer, graphics display, and power system. The prototype device's hardware is comprised of a reconfigurable radio enabling communication in multiple frequency bands, a host computer to perform signal processing and control the entire system, a touchscreen LCD and audio interface for display and user-control, and a rechargeable battery for portable operation. The block diagram in Figure 4 illustrates the system's basic architecture and depicts high-level interfaces between components within the system. In the following sections, we discuss the design and integration of each hardware component and key interfaces, highlighting the challenges encountered throughout the process.

3.1. Enclosure

We considered three options for the enclosure: sheet metal, machined aluminum, and stereolithography (SLA). SLA is the most widely used rapid-prototyping technique for producing three-dimensional parts quickly and efficiently – the process itself takes on the order of hours to complete. SLA fabrication works by laser-hardening light-sensitive plastic in consecutive cross-sectional layers of the part being fabricated, followed by minor cosmetic finishing. Although SLA can be more expensive than some alternatives, we chose to fabricate the enclosure using SLA because a more customized enclosure could be delivered as a turnkey solution in the shortest amount of time.

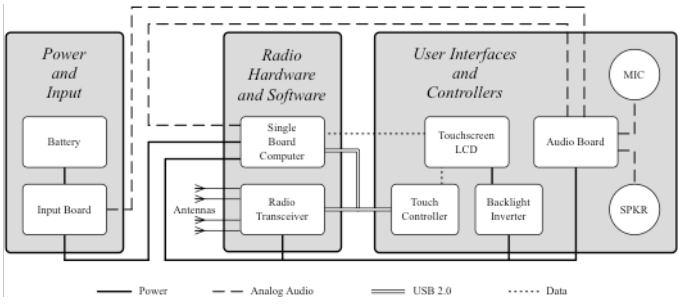

Figure 3: High-level block diagram of the prototype's major hardware elements, including power, audio, USB, and other data connectivity.

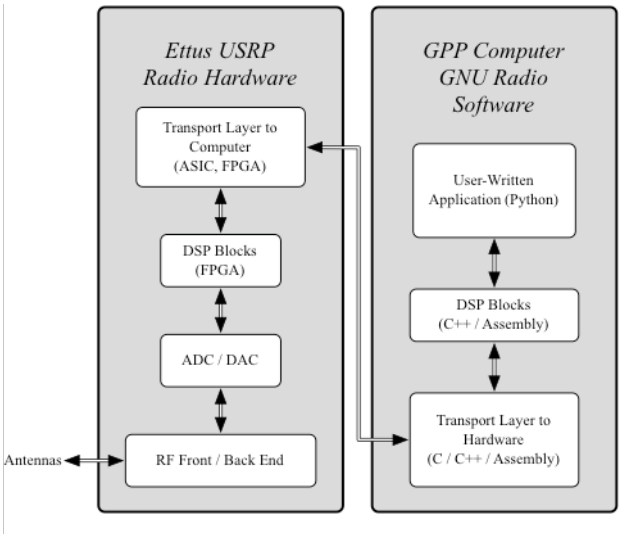

Figure 4: Block diagram depicting the functional relationship between the Ettus USRP hardware and GNU Radio software.

The enclosure design is a "clamshell" with the computer and user interface (LCD, microphone, and speaker) on the top half, and the Ettus USRP in the bottom half. Figure 5 shows the open clamshell with internal components. A grounded sheet of aluminum separates the top and bottom halves of the enclosure. This sheet helps shield electromagnetic interference (EMI) between the two halves and provides a mechanical connection for several components. The EMI shield is a critical component for reliable operation of the radio; without adequate isolation the USRP picks up too much EMI from the SBC, dramatically degrading performance. The use of a COTS single-board computer (SBC) and LCD, both discussed below, were two key factors in determining the device's form factor.

3.2. Host Computer

For the purposes of building a few prototypes, a commercially available SBC can provide flexible and powerful processing with little capital investment or development time. A number of vendors offer embedded processing boards intended for OEM integration. Taking full advantage of the existing GNU Radio software and Ettus USRP transceiver required a SBC with a high-end chipset that was in common use. A lower-end Intel Celeron processor would have been sufficient for most applications, but an Intel Core 2 Duo offered superior performance with only a modest increase in power consumption, if any, over the Celeron. We chose the Commell LS-371 SBC because it has one of the best performance-to-size ratios among the SBCs we evaluated, and it incorporates all of the peripherals we required – USB 2.0, serial graphics, and audio input / output.

The LS-371, like most modern computers, can boot from a variety of data-transport mechanism. In order to simplify the enclosure design and save space, we opted to use the compact flash (CF) memory-slot on the bottom of the board, recognizing that the throughput would likely be slower than other boot device connections. Using a stock LS-371 for testing different boot devices, the primary difference for our particular application was in boot time, as covered more in Section 4; applications, once executing, ran at roughly the same speed.

3.3. Graphics Display

The device includes a display and touchscreen interface that

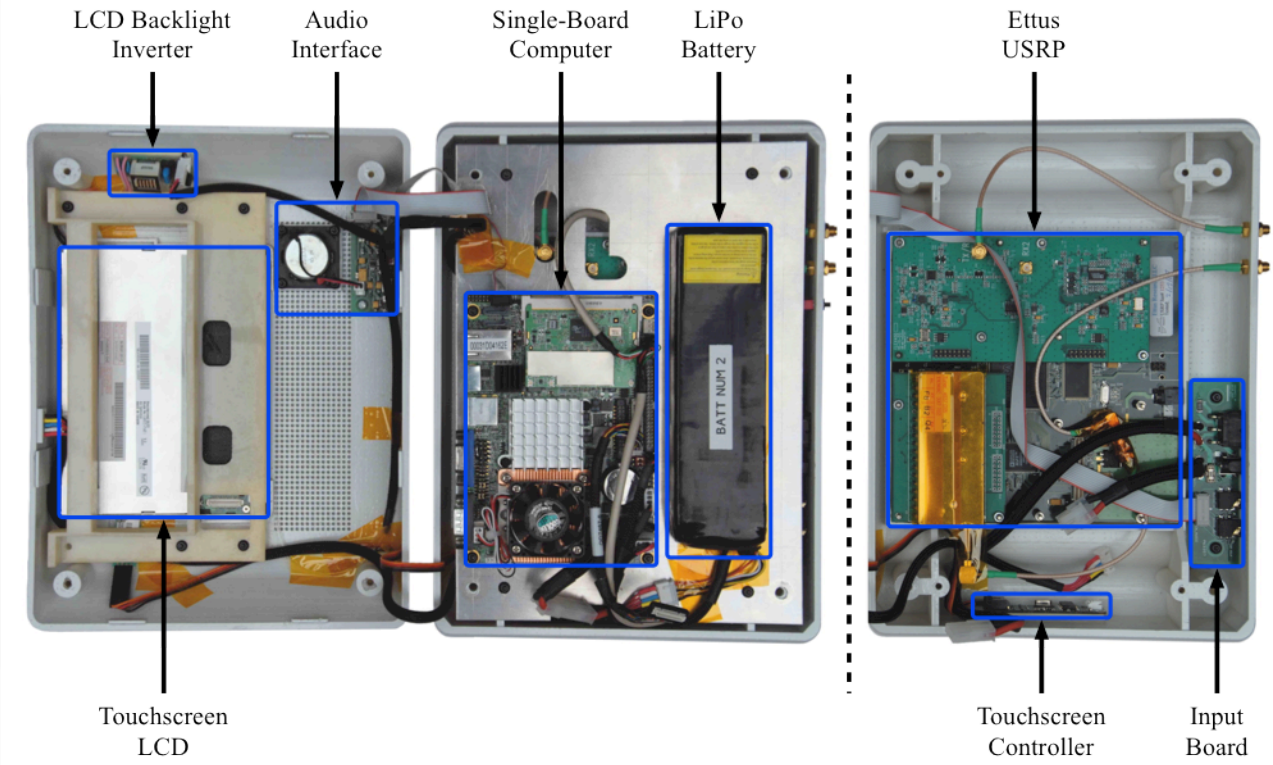

Figure 5: Internal view of the prototype showing the touchscreen LCD, backlight inverter, and audio interface on the left; the single-board computer and rechargeable battery in the middle; and, with the RF shield removed, the Ettus USRP, input board, and touchscreen controller on the right.

substantially enhances the platform's functionality. However, incorporating the graphics display was the source of several unanticipated challenges, and increased the design complexity. A simpler approach would have been to use a character LCD, but that would have limited the user interface options and made on-device development more challenging. The ideal solution was a small computer display that interfaces directly to the LS-371 and a touchscreen that emulates mouse clicks.

The main difficulties stemmed from the fact that LCDs are usually designed for a specific product, and touchscreen overlays are typically LCD-specific. Additionally, there are a variety of signaling formats used for internal video transport, further limiting what off-the-shelf display devices would work for this application. LCDs smaller than 8.4" usually have parallel TTL-level inputs, whereas many SBCs only provide video output over a high-speed serial interface using low-voltage differential signaling (LVDS). For simplicity, we chose to use the AUO G065VN01 6.5" VGA (640x480) LCD – the smallest readily available with an onboard LVDS interface. Because the G065VN01 does not have an integrated touchscreen, we incorporated a resistive touch overly that was designed for a similarly sized LCD. A touchscreen controller encodes the overlay's output and sends the encoded data to the SBC via USB. Developing the Linux software drivers for the USB controller we chose presented some additional challenges, which are further discussed in Section 4.

3.4. Power System

It was essential that the system be portable, necessitating an internal power source with enough capacity for running useful experiments in the field. However, the computationally intensive signal processing performed by the SBC and Ettus USRP requires more power than a similar hardware-based wireless device. Because weight was also of importance, heavier batteries such as lead acid were inadequate. Lithium-ion (Li-ion) batteries, and more recent successors such as lithium-polymer (LiPo) batteries offer one of the best energy-to-weight ratios and lowest self-discharge rates available today. LiPo technology offers several additional benefits over Li-ion such as improved robustness, increased energy density, and flexible housing that enable more customized form factors. These benefits led to our decision to use a LiPo battery pack (with internal protection circuitry) constructed from four 3.7 V cells, which together weigh about one pound and provide a capacity of over 6 Ah at 14.8 V.

The LS-371 provides the 5 V and 12 V power supplies needed for the Ettus USRP, LCD backlight inverter, and audio amplifier. Although using the same power supply for the radio and digital boards results in increased RF noise, the overall design is much simpler and we found this solution to be acceptable for many applications. For non-portable operation, an external power supply can be used via a standard 2.1 mm center pin DC jack on the back of the device.

3.4. Audio Interface

The easiest way to provide the necessary audio peripherals while interfacing with the LS-371 was to design a simple audio board specific to the prototype's needs. The audio board connects directly to the LS-371's audio header and is powered by the 5 V supply. It is mounted to the top-front of the enclosure and contains a built-in microphone, amplifier for the audio signal to an internal speaker, and logic for an externally accessible audio port. The audio port provides 3.5 mm stereo line input and output jacks that are automatically selected when a plug is inserted or removed. A low-noise adjustable gain amplifier can be switched in and out of the audio signal path to provide gain for low-level input signals, such as from an external electret microphone. All of these features are configured via an onboard DIP switch, allowing audio operation tailored for varied applications.

4. SOFTWARE INTEGRATION

Even with the decision to use GNU Radio software for the radio, there were a number of software issues to address including selection of the operating system for the SBC and integration of drivers for hardware interfaces. This section discusses the choices and implementation of software, and issues that arose and how they were resolved.

4.1. Operating System

In the spirit of keeping the project open-source, we focused on Linux for the host operating system. As the SBC we chose was quite new, we had to investigate several Linux distributions before one was found that functioned reliably. Among the free mainstream distributions that supported the SBC, we found that Ubuntu 6.10 offered the highest level of functionality. After choosing Ubuntu as the host operating system, we had to integrate USB-based touchscreen software and deal with boot issues created by our choice of CF storage.

4.2. Touchscreen Drivers

The kernel-space extension ("kext") for USB-based touchscreens could not provide orientation parameters for our selected touchscreen; this kext is not designed for calibration. To make use of the touchscreen, we modified the USB touchscreen kext to add user-space options for

swapping the X and Y coordinates and inverting the resulting X or Y axis – all independent of each other. For calibration of the incoming touchscreen data with the LCD, we chose the Evtouch X.Org event driver [13], as it was the first solution that compiled with minimal changes – even though at the time it was not designed specifically for Ubuntu Linux.

4.3. Boot Disk Issues

Compared with booting from an IDE hard drive, booting from CF was around 4 times slower at roughly 4 minutes. After reviewing boot logs it was clear that a direct memory access timeout was stalling the boot process. A search of the particular error in the Ubuntu web forums resulted in a fix via adding the boot parameter "ide=nodma" to the GRUB "menu.lst" file for each boot command, reducing boot time to around 2.5 minutes.

4.3. Radio Software

GNU Radio provides basic building blocks for a "simple" analog repeater as well as a "complex" HDTV receiver; users can also create their own blocks. The software package as a whole provides a framework for experimentation, testing, and evaluation of new communications protocols. GNU Radio is powerful, scalable, robust software for real-time digital signal processing.

5. CONCLUSIONS

We have successfully implemented a portable software radio prototype built primarily using commercial off-the-shelf components and open-source software. Given the ever-increasing computational power of GPPs, as well as continually increasing interest and funding for software radio and related projects, we believe that GPP-based software radio will soon provide the processing power, scalability, and reconfigurability required by current and future communications problems. Going forward, emerging applications of software radio [13] offer the possibility of revolutionizing the wireless industry through cognitive functionality [14] – allowing radios to dynamically access spectrum as needed, and moving transmissions elsewhere if legacy radios communicate using the same spectrum.

6. ACKNOWLEDGMENTS

This work has been supported in part by the US National Institute of Justice (NIJ) through grant 2006-IJ-CX-K034 and the US National Science Foundation (NSF) under grant CNS06-26595.

The authors thank Phil McPhee for mechanical engineering design work, Brynnage Design (http://brynnage.com/) for next-day SLA fabrication of the enclosure, and our Software Radio Group – including Neil Dodson, Andrew Harms, Ben Keller, Marcin Morys, and Yaakov Sloman – for their continuing efforts.

7. REFERENCES

[1] GNU Radio Website, 2008. [Online]. Available: http://www.gnuradio.org/

[2] Ettus Research LLC Website, 2008. [Online]. Available: http://www.ettus.com/

[3] CalRadio Website, 2008. [Online]. Available: http://calradio.calit2.net/

[4] High Performance SDR Website, 2008. [Online]. Available: http://hpsdr.org/

[5] KU Agile Radio Website, 2008. [Online]. Available: http://agileradio.ittc.ku.edu/

[6] Rice WARP Website, 2008. [Online]. Available: http://warp.rice.edu/

[7] Lyrtech Small-Form-Factor SDR Website, 2008. [Online]. Available: http://www.lyrtech.com/

[8] University of Texas HYDRA Website, 2008. [Online]. Available: http://users.ece.utexas.edu/~rheath/research/prototyping/mimoadhoc/

[9] Virginia Tech Chameleonic Radio Website, 2008. [Online]. Available: http://www.ece.vt.edu/swe/chamrad/

[10] Virginia Tech Cognitive Engine Website, 2008. [Online]. Available: http://www.cognitiveradio.wireless.vt.edu/

[11] Virginia Tech OSSIE Website, 2008. [Online]. Available: http://ossie.wireless.vt.edu/trac/

[12] Evtouch Website, 2008. [Online]. Available: http://www.conan.de/touchscreen/evtouch.html

[13] J. M. Chapin and W. H. Lehr, "Cognitive Radios for Dynamic Spectrum Access – The Path to Market Success for Dynamic Spectrum Access Technology," IEEE Communications Magazine, vol. 45, no. 5, pp. 96–103, May 2007.

[14] S. Haykin, "Cognitive Radio: Brain-Empowered Wireless Communications," IEEE Journal on Selected Areas in Communications, vol. 23, no. 2, pp. 201–220, February 2005.