Design of a Practical Handheld Software Radio

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

September 21, 2012

Abstract

We've stood on the precipice of the software radio revolution for many years. However, these devices still only sit in expensive commercial and military applications. State of the art radios use power-hungry SRAM based FPGAs that demand fast interconnects to host processors operating under heavy loads. These devices lack the capability of low-power operation which barrs wide scale deployment. A new architecture is presented for the software radio, using a Flash based FPGA to enable true low power operation. Duty cycling is possible since the FPGA, processor, and radio front ends can all be shut off. An integrated hard ARM Cortex-M3 on the same die as the FPGA allows for rapid and efficient communication over an on chip AHB-Lite interface. Linux runs on top of the ARM, enabling robust networking tasks from a battery operated software radio.

Keywords

software radio, low-power, embedded systems

1 Introduction

My fascination with radio started when I learned that I grew up a few miles down the road from Nikola Tesla's old Wardenclyffe Laboratory, the site that he planned to bring universal communication to the whole world with. It has come to pass that the whole world communicates now through radio. Everyone carries a Internet linked smartphone. But sadly, Tesla's lab lays shattered and forgotten, just as the *universal* component to the communication medium lays shattered and forgotten.

On a particularly chilly night in New York, I was explaining the repercussions of SOPA or ACTA. "As long as there are wires in the Internet, and people control those wires.... you're saying the Internet will be threatened by closed-speech campaigns?" my father asked. "Exactly," and that's when I realized that I wanted to build a radio, to help us break free of all the wires.

I was in luck, and I discovered two interesting facts that night as I researched how a cell phone was made. First, I found Eric Blossom's Exploring GNU Radio manifesto. The possibilities of software radio tantalized me since I had focused on VHDL design at the University of Maryland. I also learned that I could legally attempt to build a software radio as a licensed Amateur Radio operator. I scribbled down 3 main goals for my new project that night:

- To build a *software defined radio (SDR).*

- The device should be an entry level Amateur Radio. Gordon recommends a dual band handheld to be your first radio, so *small*, *cheap*, and *low-power*.

- It had to be *open hardware & software*, with no restrictions on commercial use; the success of projects like the Arduino and the MakerBot vitally depend on their permissive licensing models.

But little did I know how tall of an order I had prescribed myself! There are no existing open source low power software radios because they are power hungry. For example, the USRP draws around 10 Watts during normal operation[1], and that's not including the CPU that's loaded at 100% as GNU Radio chugs through incoming samples. I love my USRP, but strapping it on my back with an Ultrabook and hefty batteries just doesn't classify as a handheld. I had to go back to the drawing board.

And that's when this project got legs of its own. My best friend from University, Aaron Schulman, was working on his PhD at Michigan at the time. I told him my intentions to build a handheld SDR, and to my surprise, his advisor and peers had recently had a breakthrough on a sub-watt SDR. Not only that, but when I told them what I wanted to do with the technology, they wanted to open source their work with me.

2 Properties of a Low-Power SDR

This section explains the key findings in Dutta et al.[2] that makeup a low-power software radio. First off, it is a *superhet software radio*, in that a superheterodyne radio transceiver sits on one side of the analog-to-digital conversion (ADC) and digital-to-analog conversion (DAC), and on the other side is the FPGA and processor.

There are three necessary pieces to a low-power SDR: full radio duty cycling, unprecedented system integration, and extensive built-in measurement of power usage.

2.1 Full Radio Duty Cycling

Radio dominates system power budget in low-power wireless deployments. In order to extend the battery life of a radio transmitter, its transmit chain must be capable of a *full sleep* mode when not transmitting. The same goes for the receiver, but here the concept of *duty cycling* also applies. The effectiveness of a radio's duty cycling depends most heavily on the *radio startup time*. For a low-power software radio to exist, it must offer full duty cycling of both the RF front end, as well as the processor.

This is the reason that no current state of the art SDRs like the HPSDR and USRP can become low power. It's because they use SRAM based FPGAs, which make duty cycling impossible. At powerup, the configuration must be read off of the nonvolatile storage and loaded into the FPGA fabric. This causes a spike in current at startup, and an increase in radio startup time. Therefore, SRAM based FPGAs are not ideal for duty cycling and cannot be used to make a low-power receive chain.

Dutta et al. propose the use of a Flash based FPGA to overcome the shortcomings of the SRAM based FPGAs. For instance, Microsemi's SmartFusion FPGA family offers a Flash based FPGA fabric. In this case, the configuration is stored in nonvolatile memory directly on the FPGA fabric, so there is no lengthy radio startup time or rush of current. Short radio startup time means effective duty cycling is possible, and thus the SmartFusion's FPGA is ideally suited to make a low-power SDR receiver.

2.2 System Integration

The state of the art in software radios include using high speed and wide busses for modularity, but this increases both the size and cost; two main issues with a portable, low-power radio. In particular, the interconnect between FPGA and processor is critical. SDRs like the HPSDR maximize the throughput on multi-gigabit ethernet links, which demands a second computer for pro-

cessing.

In contrast, the SmartFusion includes a hard ARM Cortex-M3 on the same die as the FPGA fabric. The two parts communicate via an on chip AHB-Lite interface. This allows the processor and FPGA to pass data around quickly, with low power, and without the need of a large external bus, ultimately saving board space.

2.3 Power Measurement

As handheld radios become more and more complicated, it's becoming more and more difficult to understand how the device is consuming scarce power resources. This is why power measurement is the final key component to a low power software radio. By measuring and understanding how power is used on the digital, radio fron tend, radio synthesizer, and amplifier rails, a better understanding of how the device consumes power becomes available. Since this device is a software radio, new algorithms to lower the power even more can be deployed as a software upgrade.

The SmartFusion is well suited for the power measurement task, as it includes a full suite of current measuring circuits. The recorded data can be analyzed by both circuit and driver designers to increase power efficiency.

3 Architecture

3.1 Integrated Circuits

Realizing a cheap prototype quickly drove me to use the *Emcraft SmartFusion System-On-Module (SOM)*. The SOM integrates a SmartFusion A2F500, containing both an ARM Cortex-M3, 500K system gates; with an external 16 mbytes PSRAM, and 8 mbytes NOR Flash. This mini-board runs the Linux 2.6 kernel with busybox, an embedded systems shell kit. It exposes 2 UARTs, 2 SPIs, 2 I2Cs, 32 GPIOs to the FPGA, 16 GPIOs to the Cortex-M3, and an Ethernet MAC Physical Layer. All using a compact 80-pin bus to communicate with the daughterboard.

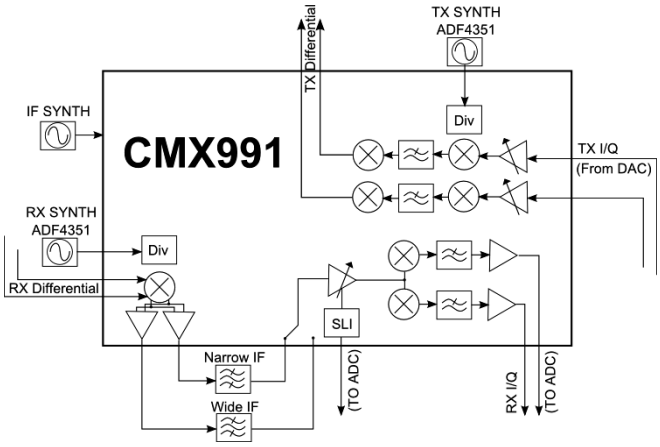

Figure 1: The CMX991 is a full I/Q Transceiver that operates from 100 MHz to 1GHz.

This device completely abstracted away the complexities of design and fabrication of a 10-layer mixed signal circuit board with BGA components. Weighing in at 30mm×57mm, about the size of two postage stamps and conveniently the width of an iPhone, it helped drive innovation. Oh, and there's a full GCC toolchain available for free.

The radio frontend, the *CMX991 from CML Microsystems, Ltd.* is a full superheterodyne I/Q Transceiver from 100MHz to 1GHz on a chip. It was chosen for its coverage of the 2m and 70cm bands. As shown in Figure 1, a receive mixer, IF variable gain amplifier, IF Signal Level Indicator (SLI); and a baseband I/Q demodulator with differential outputs is included. A transmit I/Q modulator, IF mixer, and image-reject-upconverter come as well. It has up to 2MHz of bandwidth.

The ADCs and DAC were chosen for their low power. The I/Q receive signal is sampled by the *Analog Devices AD9288*, a dual 8-bit, 40 MS/s ADC; though upto a 10-bit, 100 MS/s pin-compatible chip is available. The I/Q transmit signal is generated by the *Maxim MAX5189*, a dual 8-bit, 40 MHz DAC; though a pin-compatible 10-bit version is also available. Automatic gain control is adjusted from the FPGA via sampling the CMX991's SLI with a *National Semiconductor ADC081S101.*

3.2 Synthesizers

Due to the superheterodyne nature of the device, both radio frequency and intermediate frequency synthesizers are required for operation. Again, the ability to shutoff the synths, power up, and lock in a short timeframe is critical in the application of a low power radio.

For the radio frequency synthesizer, a wideband *Analog Devices ADF4351* has been chosen, the same synthesizer used in the URSP WBX daughterboard. This is a very wideband synth, operating from 35 MHz to the multi-gigahertz range, and is more than enough to drive the CMX991. It also includes a low-power mode where it draws microamperes, and a built in lock-detect. This enables the ADF4351 to be shutoff when not transmitting or receiving, and then quickly enabled and locked for stable radio operation.

The intermediate frequency synthesizer operates at 180 MHz, divide-by-2 and divide-by-4 circuitry allow for 90 MHz and 45 MHz intermediate frequencies, respectively. The CMX991's SPI bus controls power to this circuit, so it can be shutdown during sleep times.

3.3 Power

The prototype device takes a 5V DC supply voltage and is rated to draw up to an amp during transmit or receive. A 3.3V LDO regulates the supply to the digital components. A ferrite bead separates the digital from analog voltages, which provides stability to the radio frontend and sensitive ADC/DAC analog parts. Headers are available on the various voltage rails to try out switching regulators and LiPo charging circuits.

3.4 ARM-FPGA Interface

The efficiency of Dutta et al.'s design is most apparent in how they configured the FPGA's Verilog. Two main components drive this efficiency, *radio_ctl*and *fifo_ctl*, which can be seen in in Figure 2 along with the rest of the SmartFusion stack.

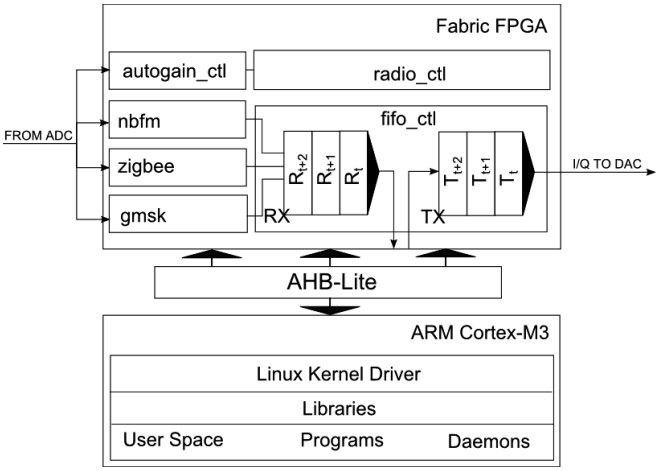

Figure 2: Application of SmartFusion. Note that the FPGA and ARM are on the same die.

The *radio_ctl* block looks like a small memory block to the processor, but contains all of the necessary bus-control logic to manipulate the radio front end and synthesizers. For instance, frequency can be changed by setting the desired frequency in a block of memory. This write initiates an AHB-Lite transfer which triggers the ADF4351's SPI bus sequence to dial the synthesizer to the correct frequency. Control data can also be read from the radio, for instance, the power status of the radio front end can be read back in over the SPI bus and relayed to the processor.

The *fifo_ctl* looks like a character device, but internally contains asynchronous transmit and receive queues. It is the interface for shuttling actual data from the radio, through the AHB-Lite interface, to the processor, and vice versa. Each item in the queue is a command and optional packet. For instance, when a user space application writes data to the radio driver, it is packetized and enqueued in the fifo_ctl transmit queue. The most recently enqueued packet in figure 2 is T_{t+2}. If the radio needs to be turned on to send the data, fifo_ctl issues the necessary commands to the radio_ctl component. When the transmitter has stabilized, the packet is I/Q modulated and sent through the DAC to the ra-

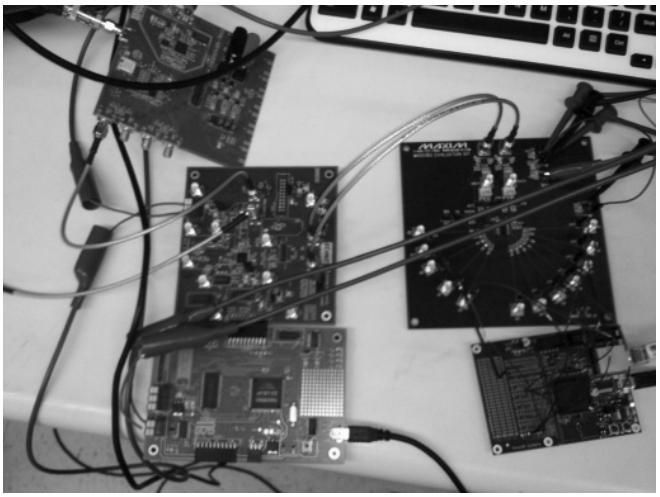

Figure 3: The transmit rig setup as a number of dev boards hooked into power supplies. From lower right clockwise the boards are, SmartFusion SOM, CMX991, ADF4351, and MAX8351.

dio for transmit.

The fifo_ctl really shines during receive, as the processor doesn't even need to be turned on until a full packet has been received. Imagine the radio is currently in sleep state. When the duty cycle timer goes off, the radio receive chain is powered up. The I/Q samples come off of the DAC and into the FPGA, and the data is correlated for an incoming signal. Lets say a simple magnitude-based AM-squelch has been inserted. If the squelch is triggered, then the incoming voice data is enqueued in the fifo_ctl receive queue. Data on this queue causes an interrupt in the processor, which wakes up the kernel driver.

4 First AM Voice Contact

The first AM Voice transmit was sent from the proof of concept on July 11, 2012 in the WIESEL lab at the University of Utah. It was transmitted on the 70cm band and received by a USRP. The proof of concept can be seen in figure 3. The reverse, transmit from USRP and received by the device occurred a day later. This section will describe the flow of that first transmit.

On reset the transmit queue is cleared on the FPGA, which puts the entire transmit chain in sleep mode. A user-space program running inside of the SOM's Linux kernel packetizes the raw amplitude data of a voice recording, and enqueues it on the transmit queue in the FPGA. When the queue becomes non-empty, the transmit chain powers up from sleep state. The FPGA now sets the correct frequency, and waits for the transmit and intermediate frequency synthesizers to lock.

When the local oscillators have locked, the transmitter is turned on and the samples are fed into the DAC from the FPGA. The DAC converts the digital I/Q signals into analog, which is fed into the radio front end. The front end mixes to the IF, and then image-reject-up-converts to the transmitting frequency. The signal feeds through the transmit power amplifier, and is sent out on the antenna.

5 Discussion

This paper presented the architecture of a handheld, superhet software radio. It has been argued that there are three main components to a low power SDR: the ability to fully sleep and duty cycle, unparalleled system integration, and complete power measurement.

Current popular SRAM based FPGAs are not capable of duty cycling, and require a host processor, therefore they are ill-suited for low power designs. Flash based FPGAs are explored, which enable a low-power SDR.

The ultimate goal of this project is to open source the design, schematics, Verilog, and kernel drivers, as well as to small-scale manufacture the device as a kit. The deployment of low-power SDRs will hopefully hold many advancements in radio, including techniques to alleviate the spectrum crunch problem. One thing is for sure, Amatuer Radio Operators are uniquely positioned to continue advancing the art and science of radio.

References

[1] GNU Radio - USRP General FAQ. (2010, 3 12). Retrieved from http://gnuradio.org/redmine/projects/gnuradio/wiki/UsrpFAQGen

[2] Dutta, P., Kuo, Y., Ledeczi, A., Schmid, T., & Volgyesi, P. (2010). Putting the software radio on a low-calorie diet. Hotnets-IX Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Retrieved from http://dl.acm.org/citation.cfm?id=1868467