Reconfiguring the Software Radio to Improve Power, Price, and Portability

Ye-Sheng Kuo[†], Pat Pannuto[†], Thomas Schmid[‡], and Prabal Dutta[†]

[†]Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, MI 48109
{samkuo,ppannuto,prabal}@eecs.umich.edu

[‡]Electrical and Computer Engineering Department
University of Utah
Salt Lake City, UT 84112
thomas.schmid@utah.edu

Abstract

Most modern software-defined radios are large, expensive, and power-hungry, and this diminishes their utility in low-power, size-constrained settings like sensor networks and mobile computing. We explore the viability of scaling down the software radio in size, cost, and power, and show that an index card-sized, sub-$150, 'AA' battery-powered system is possible using off-the-shelf components. Key to our approach is that we leverage an integrated, reconfigurable, flash-based FPGA with a hard ARM Cortex-M3 microprocessor which simultaneously enables lower power and tighter hardware/software integration than prior designs. This architecture allows us to implement timing-critical MAC protocols and validate the speculated performance of several recent MAC/PHY primitives and protocols including Backcast, A-MAC, and Glossy using an IEEE 802.15.4-compliant radio implementation that interoperates with commercial radios. The work also identifies several enhancements in the underlying hardware components that could improve power, performance, and flexibility.

Categories and Subject Descriptors

B.4.1 [**HARDWARE**]: Input/Output and Data Communications—*Input/Output Devices*; C.3 [**COMPUTER SYSTEMS ORGANIZATION**]: Special-Purpose and Application-Based Systems

General Terms

Design, Experimentation, Measurement, Performance

Keywords

Software-Defined Radio, IEEE 802.15.4

SenSys'12, November 6–9, 2012, Toronto, ON, Canada.

1 Introduction

Partridge predicts that by 2020, every commercial and military radio will be a software-defined radio (SDR) system [25], with fixed functions replaced by programmable or reconfigurable components, as Figure 1 shows. The military is well on its way with the Joint Tactical Radio System (JTRS [1, 19]). JTRS is the next generation voice and data communication system for the U.S. military and is based on the Software Communications Architecture (SCA [13]), an open-architecture describing how software and hardware works together. Commercial radios have been slower to adopt the advantages of SDR systems. GSM base stations are some of the first commercially available SDR solutions [23]. The rapid innovation in mobile phone radio standards places a major burden on the network providers, as hardware has to be regularly upgraded. SDR base stations alleviate this problem with firmware upgrades that support new protocols.

More general SDR architectures offer high performance and are optimized for flexibility [3, 9, 10, 14, 18, 31]. This has led to large, expensive, and power-hungry systems that draw between a few and hundreds of Watts, and cost over $1,000 per radio. But these developments run counter to the growth in small, portable, and battery-powered wireless systems. If Partridge's prediction should hold, SDR systems must slim down and start to explore the mobile space.

Prior research has proposed one path for putting the SDR on a "low-calorie" diet, identifying four key requirements to improve power, price, and portability [7]:

Radio Duty-Cycling. The radio dominates the power budget in low-power wireless deployments [29, 30] so radio duty-cycling and accurate time keeping are key [5].

Low-Power FPGA. SDRs distribute their functionality across many components, so both the radio frontend *and* the processing backend (FPGA, GPP, or DSP) must be jointly power-optimized.

System Integration. Built from modular components, SDR systems are built to be flexible, but this modularity inherently increases cost and size.

Measurement. The main goal of a mobile, battery-operated SDR system is low-power operation. Thus, such a system should integrate energy metering tools, exposing performance metrics for hardware and software.

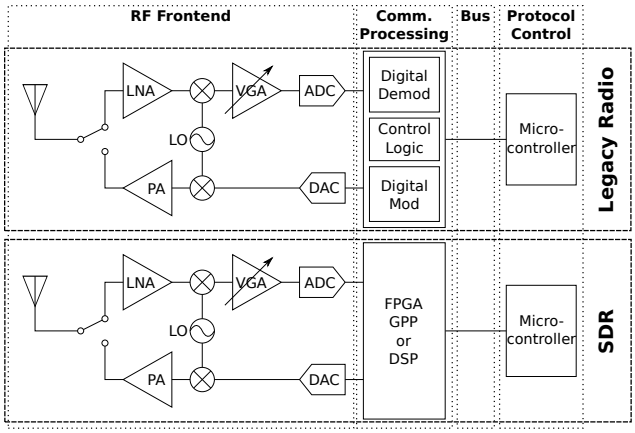


Figure 1: Architecture of standard and software-defined radios. The main difference is the fixed communication processing in a standard radio vs. the reconfigurable/programmable processing possible in a software-defined radio, which enables rich experimentation and flexibility.

This work presents μSDR, the first realization of a recently proposed SDR architecture [7]. The goal of this work is to explore the veracity of the claims made in the literature, to see how close we can get with current technologies, and to identify the open problems and hardware improvements that remain to realize the vision of ubiquitous software radios. An obvious question is whether it is possible to refashion any of the existing SDR platforms into one that is small, inexpensive, and low-power. We show that none of the current SDRs can achieve low-power operation due to their particular architectural choices in processing, interconnect, and power control and management.

The design of μSDR addresses the four requirements by choosing a flash-based FPGA and highly integrated components, thus reducing size, cost, and power. In particular, flash-based FPGAs can be duty-cycled, whereas traditional SRAM-based ones cannot. μSDR also relaxes some of the flexibility that other SDR platforms provide, and integrates the RF frontend, baseband processing, and application processor all on one PCB board. Therefore, the RF frontend is fixed for a single radio band. The advantage of this approach is the smaller component count as no PCB interconnects are necessary. Fine-grained power control and a temperature-compensated crystal oscillator provide key elements for low-power operation. In addition, the physical size reduces significantly, as no space has to be allocated for potential future expansion boards or a modular interconnect. The higher integration also reduces the manufacturing cost, resulting in a final cost that is less than $150.

2 Related Work

While there are a diverse array of SDR platforms [3, 14, 18, 31], none of the current platforms are suitable for low-power radio research and development. Table 1 shows a number of prior SDR platforms. In contrast, μSDR is optimized for power, cost, and portability, without appreciably sacrificing flexibility or usability.

2.1 Throughput and Latency

A strongly limiting factor for any SDR system is the available throughput and latency of the interconnect between the "hard" and "soft" portions of the SDR platform [28]. As this latency defines the critical path for the control loop, previous work places significant focus on minimizing it.

SORA uses the PCI-Express bus for its bounded latency (sub 1 μs) and high throughput (up to 16 Gb/s), at the cost of high power and a PC form factor for the platform. This design affords SORA the support of a complete PC operating system for multi-tasking and control, at the cost of run-time adaptivity. USRP also relies on the PC operating system for much of its command and control functionality, however it eschews the fixed form-factor of the PCI-E bus, instead relying on more conventional peripheral buses: USB (480 Mb/s) or Ethernet (1 Gb/s). While these buses do provide high throughput, Nychis et al. find their latency to be highly variable, as high as 9000 μs [22]. For precise protocol timing, such high and variable latency is untenable.

μSDR, in contrast, follows in the footsteps of SODA, tightly coupling the hardware compute engine with an ARM Cortex-M3 core [14]. Both systems allow for low latency (0.46 μs) and high throughput (172 MB/s). Neither SODA nor μSDR use an operating system in the traditional sense, however as a custom micro architecture SODA goes even further. It defines a slightly customized VLIW+SIMD ISA and sacrifices traditional memory consistency semantics for performance. Users of SODA then must carefully hand-tune their programs not only to optimize performance, but to even run correctly. SODA is not a full-featured SDR platform, rather an advanced DSP architecture. While this microarchitectural compromise allows for better performance per watt, μSDR has been realized in existing commodity hardware (as opposed to just simulation), with a measured full system power draw only slightly higher than SODA's theoretical optimized form, as Table 1 shows.

2.2 Power

Before the discussion of power and portability, we first divide existing SDR platforms into two broad categories: those built on a PC or PC-architecture and those that are stand alone. Power and portability are not a design consideration for these PC-platforms (SORA and USRP 2), thus it is unsurprising that μSDR, along with the other embedded platforms, are orders of magnitude better in these metrics.

Of the remaining platforms, KUAR, WARP, USRP E100, and SODA, μSDR excels in power draw and is competitive in size with all except the custom chip SODA. Both KUAR and the USRP E100 can be considered "near-PC" platforms that embed low-power PC-like components in a compact form factor. KUAR does not explicitly report power numbers, but builds a custom board driven by a Pentium M whose most efficient model draws 5 W [12], which we use as a generously conservative approximation for its power draw. The USRP E100 datasheet reports 15 W load with RF daughtercard installed. In a our own power measurements, shown in more detail in Section 5, we find an idle power draw of 5.5 W. The WARP base system draws about 10 W, but is capable of supplying up to 30 W to each of four daughter cards, allowing for a peak power draw of 130 W.

Platform	Sleep	Power	Size	Interconnect	Throughput	Cost	Realization
SORA [31]	-	>100 W[c]	36000 cm^3[3c]	PCI-Express	16.7 Gb/s	$2000[c]	Research
KUAR [18]	-	>5 W	240 cm^3[3f]	PCI-Express	2 Gb/s	-	Research
SODA (180 nm) [14]	-	~3 W	26.6 mm^3[3e]	DMA[a]	24 Mb/s[b]	-[d]	Simulated
SODA (90 nm) [14]	-	~0.5 W	6.7 mm^3[3e]	DMA[a]	24 Mb/s[b]	-[d]	Theoretical
WARP [3]	-	10~130 W	800 cm^3[3f]	Parallel MGTs[g]	24 Gb/s	$9750	Research
USRP 2[c] [10]	-	7.9~13.8 W[c]	1760 cm^3[3c]	Ethernet	1 Gb/s	$1700[ci]	Commercial
USRP E100 [9]	5.5 W	9~15 W	1760 cm^3	OMAP 3 GPMC	1.3 Gb/s	$1300[j]	Commercial
μSDR (this work)	0.32 W	1.4 W	203 cm^3	AMBA	1.4 Gb/s	$150[h]	Research

[a] Memory bus architecture not specified [14].
[b] Inferred minimum, may be faster.
[c] Requires a companion PC. Not factored in power, portability, or cost for the USRP 1/2.
[d] SODA is a custom chip that would likely have an extremely high die cost, but low per-unit cost.
[e] Assumes 1 mm thick.
[f] Assumes 2 cm thick.
[g] "Multi-Gigabit Transceiver", an interconnect technology built into Xilinx FPGAs. Uses up to 8 parallel 3 Gb/s transceivers
[h] Assumes 1,000 unit production run. See Table 2 for detailed breakdown.
[i] Estimated material cost approximates $400.
[j] Estimated material cost approximates $360.

Table 1: A comparison of SDR platforms. The range in power comes from boards whose power usage varies depending on the presence and type of daughter card installed in the system. Where possible a measured idle / sleep power is also shown. For platforms that only list area we make reasonable assumptions on height. μSDR is 10% the cost of the next most expensive SDR platform, yet provides parable speeds in the smallest non-IC package. It uses less power than any realized hardware and nearly ties the previous best theoretical hardware.

All of these previous systems have one thing in common: their power usage is reported in *Watts*. With μSDR, we deliver the first realized *sub-Watt* SDR platform – a significant milestone. μSDR is the first SDR platform capable of running for a full day on a pack of AA batteries [1]

2.3 Portability

As we enter the era of *sub-Watt* SDR platforms, we find that the size, and in turn portability, of the SDR platform becomes a critical design consideration. The current μSDR design is approximately four times the size of popular research motes such as the Mica, TelosB, or Tmote Sky, shown in Figure 3. We note that with the removal of the external memory controller, μSDR's extra interfaces (such as Ethernet), and a slightly more compact layout, the μSDR platform could easily be made half its current size.

2.4 Cost

Perhaps the greatest advantage of the μSDR system compared to previous work is its extremely low cost – an order of magnitude less expensive than other SDR platforms. The first large cost reduction comes from building a standalone platform, rather than relying on a costly support PC (SORA, USRP 2) or the slightly less costly embedded PC-like environment (KUAR, USRP E100). The next biggest saving comes from eschewing the more traditional daughtercard based SDR approach, instead opting to design a dedicated 2.4 MHz RF frontend.

[1]Extrapolating from [26], we consider four AA batteries with average of 2500 mAh and a fully charged nominal voltage of 1.6 V . The minimum supply voltage to μSDR is 4 V, yielding 9.75 Wh of usable power. This is enough energy for μSDR to run 7.5 hours at full power or 30.5 hours in maximum sleep.

While the μSDR system lacks physical daughtercards, it does not completely sacrifice modularity. Separate RF frontends can be "dropped in" to the schematic and used interchangeably. Third parties are already designing a new board with a 5 GHz frontend. We also find considerable cost-savings by using fewer and less-powerful FPGAs than the WARP platform. Despite the significantly lower processing power of the μSDR platform, it offers sufficient performance to support an IEEE 802.15.4 standards-compliant radio on a significantly smaller budget than other SDR platforms.

2.5 Deployability

Ultimately, μSDR contributes what we argue to be the first truly *deployable* SDR platform. Previous platforms were tied to PCs (SORA, USRP 2), tied to large power supplies (KUAR, WARP, USRP E100), or existed only as simulations (SODA). Even resolving these issues, all prior platforms were prohibitively expensive for any large scale deployment. We estimate the cost per node of a 1,000 node μSDR deployment to be only $150, a full order of magnitude better than the previous state of the art.

3 Design and Implementation

This section presents the key architectural and design decisions that underpin the μSDR platform. Three main designs goals guide the system architecture: *low-power*, *small size*, and *low cost*. We relax the modular design of existing SDR systems to reduce size and cost, while promoting the FPGA to a first class citizen.

A fast interconnect is necessary for meeting the tight timing and low radio latencies that modern radio standards require. Hence, we must use a fast, but low-power interconnect between the FPGA and the processing core to achieve this.

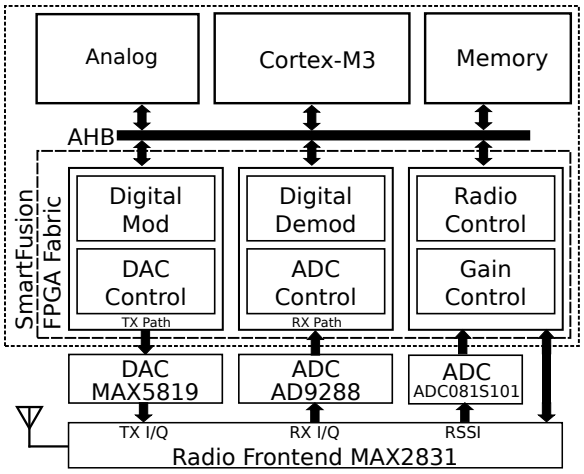


Figure 2: μSDR system architecture. The FPGA fabric hosts the transmit, receive, and radio control, and is directly connected to the AMBA High-performance Bus (AHB), together with the memory, analog compute engine (ACE), and the ARM Cortex-M3 core. This interconnect allows direct memory-mapped access from the core to the SDR blocks.

One possibility is to use an external memory interface existing on many low-power microcontrollers (MCUs) to connect to the FPGA. While this allows for a flexible choice of MCU and FPGA, it has the disadvantage of a larger footprint, constrained I/O bandwidth, and potentially increased costs.

ARM provides several options both on the low-power computing core and bus architecture front. All major FPGA vendors sell SoC packages that contain high-speed ARM cores integrated with leading-edge FPGAs. The cores and the FPGA are most often connected through the AMBA High-performance Bus (AHB). The AHB is a pipelined, single clock-edge on-chip bus to connect peripherals, memories, and cores on a SoC and provides a bandwidth of up to 16 GBit/s. An even faster version of the AHB is the Advanced eXtensible Interface (AXI) which is used in Xilinx's Zynq reconfigurable SoC that marries a dual-core ARM Cortex-A9 with a Xilinx FPGA in one package [33]. Altera has a similar SoC combining a dual-core ARM Cortex-A9 with their Arria V or Cyclone V FPGA series. Altera's bus interface has a peak bandwidth of up to 100 GBit/s [2]. While these systems offer impressive performance, their power draw is measured in Watts, making them unsuitable for a low-power, battery-operated solution.

Microsemi, an FPGA vendor with a low-power, flash-based product line offers a third alternative: the SmartFusion customizable System-on-Chip (cSoC) [17]. SmartFusion contains an ARM Cortex-M3 with a flash-based FPGA, connected through an AHB bus interface. The SmartFusion offers instant-on at power up, as configuration is stored in the flash-based FPGA. The AHB interconnect allows developers to write custom peripherals on the FPGA, and access them via memory-mapped I/O, as if they were a regular peripheral. This architecture allows for fast and simple interaction between the software and hardware elements of SDR platform, and a very flexible software-hardware boundary.

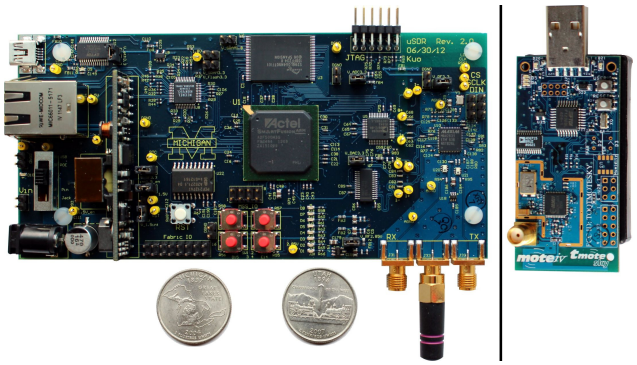


Figure 3: The μSDR platform. It measures 14.5 $\times$ 7 cm and runs for 7.5 hours without duty cycling from 4 AA batteries, allowing for the first time true mobile SDR experiments. Shown for scaling is the Tmote Sky. The current μSDR is approximately four times the size of the Sky. Removing extraneous inputs (Ethernet, etc.) and debugging headers, μSDR is approximately twice as large as the Sky.

The SmartFusion also offers a rich set of standard MCU peripherals (timers, serial interfaces, memories, etc.) and an analog compute engine (ACE) with an ADC, DAC, and amplification/filtering options. Adding a radio frontend would turn the SmartFusion into an SDR platform. However, the ADC and DAC are limited to $\sim$700 KS/s, which would only support low-rate, non-standard protocols.

We add an external ADC and DAC to overcome the sampling rate limitations of the ACE. Figure 2 shows the major components on the SmartFusion, as well as the external ADC, DAC, and radio. Figure 3 shows the μSDR platform. The SmartFusion (U1) is the large chip in the center of the PCB. The ADC (U5) and DAC (U7) are to the right of the SmartFusion. The RF section occupies the right-hand side of the board, with the radio (U10) in the middle.

3.1 Hardware Design Details

This section describes the specific chips used in the μSDR platform, their features, and the role they play.

We chose the largest SmartFusion currently available, the A2F500, which offers 512 kbyte of Flash and 64 kbyte of SRAM. The Cortex-M3 can run at up-to 100 MHz, and the FPGA provides 500,000 system gates and 24 dedicated RAM blocks (at 4608 bits each), which is sufficient for many communications processing tasks or complex custom logic.

The RF frontend is based on a Maxim MAX2831 RF transceiver chip operating in the 2.4 GHz ISM band. This RF transceiver combines all necessary RF blocks reducing necessary external components and thus size and cost. This includes a power amplifier (PA), RF-baseband converters, voltage controlled oscillator (VCO), frequency synthesizer, and baseband control interface. The frequency synthesizer supports step increments of 20 Hz, and the digitally tuned oscillator allows the MAX2831 to use a low-cost external crystal oscillator. The MAX2831 also has a rich control interface using a serial peripheral interface (SPI) as well as some dedicated I/O and analog lines for time-critical operations, such as gain control and received signal strength indicator (RSSI).

Desc	Part number	Size (mm)	Cost (1k)
FPGA	Microsemi A2F500M3G	17 x 17	$45
Radio	Maxim MAX2831	7 x 7	$4
ADC	ADI AD9288	9 x 9	$4
DAC	Maxim MAX5189	6 x 10	$5
Discretes			$34
PCB	6 layers	145 x 70	$25
Assembly			$20
Total			$137

Table 2: Cost breakdown of the μSDR platform.

The external high-speed ADC and DAC are the Analog Devices AD9288 [4] and the Maxim MAX5189 [15]. The AD9288 is a high performance dual-channel analog to digital converter supporting up to 100 MS/s and draws less than 90 mW per channel. The requirement for only a single power supply simplifies the hardware design, and a power-down mode feature is necessary for its low-power operation in duty-cycled applications. The MAX5189 is a low cost dual 8-bit digital to analog converter, operating at up to 40 MS/s. In shutdown, the MAX5189 draws only 5 μW.

In addition to the high performance parallel ADCs and DACs, a second serial ADC, an ADC081S101 [32], reads the analog RSSI output of the RF transceiver chip at rates up to 1 MS/s. This data is available on the FPGA to support automatic gain control algorithms for the LNA and VGA of the RF transceiver.

3.2 Cost Breakdown

Cost is a critical design aspect of the μSDR platform, necessary to build larger testbeds and deployments of an SDR system. Table 2 shows a cost breakdown of the major μSDR components. Even at low volumes (1k) the μSDR platform should cost less than $150, including assembly and PCB.

4 Applications

This section presents the viability of using the μSDR platform to communicate with standards-compliant, low-power radios. We implemented IEEE 802.15.4, a non-trivial physical and link layer standard used in many short-range, low-power radios.

The short-range, low-power wireless standard IEEE 802.15.4 describes the physical layer and frame structure, leaving higher layers up to the developer to design. IEEE 802.15.4 uses spread spectrum technology to increase its resilience towards channel noise. Every 4-bit data sequence is mapped to a 32-bit long chip sequence. This chip sequence is then modulated as an offset quadrature phase-shift keying (O-QPSK) signal with half-sine pulses. The sixteen 32-bit chip sequences are chosen so their cross-correlation is minimized. This scheme allows a receiver to compare incoming chip sequences to one of the sixteen existing sequences, allowing for errors within the received sequence, but resulting in a successful sequence detection with high probability.

Figure 2 shows the general μSDR architecture. For our IEEE 802.15.4 implementation, we implemented a transmitter with O-QPSK modulator representing the transmit path. For the receiver we exploit the fact that O-QPSK with half-sine pulse shapes is equivalent to minimum shift keying (MSK), and thus we use an FM demodulator. The following sections go into details of the specific implementations. To exemplify the potential of the μSDR platform, we describe in Section 4.6 enhancements we performed that usually are not found on commercial radios, but support recent advances in wireless protocol designs.

Component	Area	Max Freq.
TX	20%	100 MHz
RX	72%	80 MHz

Area optimized for minimum required frequency of 16 MHz.
Speed optimized increases area to 78%.

Table 3: FPGA size and corresponding maximum frequency. The area with respect to the 500K gate SmartFusion A2F500. The transmitter is limited by 80 MHz DAC, while the receiver is FPGA limited.

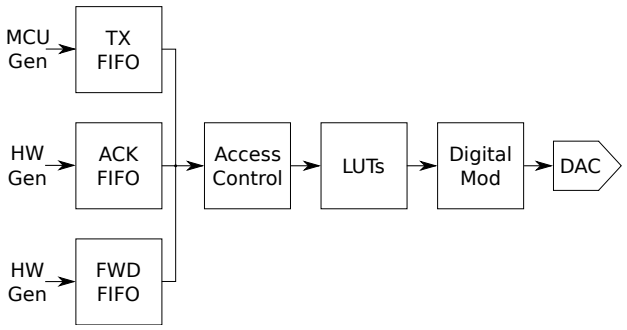


Figure 4: IEEE 802.15.4 transmitter. Three input FIFOs feed the modulator implemented as lookup tables for byte to chip, and chip to half-sine pulse mapping. This provides a compact O-QPSK implementation with support of automatic ACK generation and fast packet forwarding mechanisms.

4.1 Transmitter

We implemented the full transceiver chain as a memory-mapped peripheral on the FPGA. This reduces the burden and complexity of the software. To send out a message, the core transfers the packet from memory into a transmit FIFO (see Figure 4). The processor can start the transmission before the full packet is uploaded by sending a transmit command to the peripheral. The peripheral starts encoding the packet, starting with the preamble, SFD, and FCF, before adding the bytes from the transmit FIFO. The bytes go through several encoding steps, translating from half-bytes (4-bit) to chip sequence (32-bit), and finally they are mapped to an O-QPSK signal with half-sine pulse shapes on the in-phase and quadrature band. All these translations are done by using lookup tables, simplifying the implementation while providing high-speed processing.

The transmit FIFO shares the transmit chain with two other FIFOs for automatic acknowledgment generation and rapid message forwarding. An access control logic grants permission to a FIFO if the FIFO has data to transmit, and the transmit chain is currently empty. Section 4.4 and 4.6.2 will go into more details on these two FIFOs.

4.2 Receiver

The implemented receiver chain (see Figure 5) resembles the GNU Radio IEEE 802.15.4 receiver implementation described by Schmid [27]. It exploits the fact that an O-QPSK modulated signal with half-sine pulse shapes is equivalent to a minimum-shift keying (MSK) signal, and thus a simpler frequency shift keying (FSK) demodulator can be used to extract the chip sequence from the data stream [21].

The FSK-demodulator uses the in-phase and quadrature components coming from the RF chip at 16 MS/s to produce an amplitude modulated (AM) signal, where the amplitude corresponds to the frequency of the incoming signal. A matched filter then integrates the AM signal. The filter is sampled at the chip-frequency of 2 MS/s, indicating a logic 1 or 0. Next, a preamble detection mechanism searches for a 0x00 (a chip sequence of 64-bits) to identify any incoming packets. It also performs clock synchronization by occasionally adjusting the matched filter.

The receiver then calculates the Hamming distance D between detected symbols and the known chip sequences. The decoded symbol is the index of the sequence that has the minimum distance D from the sampled sequence. The smaller the distance D from a known chip sequence, the higher the probability for a correct decoding. Once the symbol is decoded, the receiver stores the decoded symbol in an RX FIFO and performs the frame filtering. Various interrupts (detection of the start of frame delimiter (SFD), Length, Packet completed, etc.) are generated during the frame filtering, informing the processor of the different events happening in the decoder. These interrupts are crucial for timely processing inside the software implementation of a MAC protocol.

Once a packet is fully received (e.g., when the processor receives the Packet Completed interrupt), the processor copies the packet from the RX FIFO using a DMA transfer into RAM for further processing.

4.3 Radio and Automatic Gain Control

A flexible SDR system needs fast and extensive control over the radio frontend in order to be agile and adapt to a changing RF environment. The MAX2831 provides a 3-wire SPI interface for configuration, a 7-bit parallel bus for only gain control, and an analog output providing the received signal strength. An additional I/O line switches the frontend from receive to transmit (the chip cannot be used for full-duplex radio communication).

We offload as much of the radio control into the FPGA as possible. This simplifies the software to memory-mapped operations instead of dealing with a stand alone SPI interface and incurring potential software interrupt latencies. In addition, it speeds up the control loop and allows us to send commands to the radio within 2.94 μs. For example, the MAX2831 does not provide an integrated automatic gain controller (AGC). Thus, the FPGA has to measure the RSSI, and adjust the radio gain through the SPI interface. The AGC loop tunes the gain setting within 5 μs without having to involve the processor. This high speed AGC keeps tuning the gain setting to prevent abrupt changes in signal strength and maintains a constant signal amplitude.

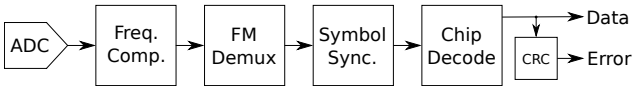


Figure 5: IEEE 802.15.4 receiver implementation. Our implementation exploits the fact that O-QPSK modulation can be decoded as MSK with adaptation of the chip sequences. This simplifies the decoder implementation.

4.4 Hardware ACK Generation

Many wireless protocols rely on acknowledgment (ACK) messages to ensure reliable communications. According to IEEE 802.15.4, if a receiver receives a packet requesting an ACK, it has to generate an ACK frame and respond to the transmitter with a precise 192 μs latency. Meeting this timing constraint is difficult for an SDR platform if it relies on the processor to software decode the signal. Pure software decoding is favorable from a software engineering perspective, as more resources and code libraries are available. In addition, the code can be written in a higher level language, such as C/C++ or Python. However, software decoding platforms must have enough computing resources and low-latency interfaces between the processor and radio to meet the stringent timing constraints. In most SDR systems, the interface between the processor and the radio frontend is considered the data bottleneck.

Meeting those strict timing constraints has become a challenge for SDR platforms. Nychis et al. characterize various latencies from the Linux kernel to the Universal Software Radio Peripheral's FPGA [22]. Their results showed that it can take up to 9,000 μs with a mean latency of 612 μs to get data from the Linux kernel to the USRP. Because of this nondeterministic latency, the USRP is ill-suited for any timing-constraint communication protocol. Nychis proposed several basic RF blocks that if implemented close to the radio, can alleviate this problem. However, it makes the system architecture significantly more complex to use as this partition must be carefully planned.

MSR's SORA solves the interface issue by using a high bandwidth and low-latency PCI-E interface, which achieves 360 ns one-way delay, but at the expense of high power. Additionally, a powerful multi-core processor with dedicated processing cores allows real-time decoding and fast replies.

In contrast, the μSDR design uses hardware decoding, i.e., demodulation and frame filtering are all performed in the FPGA. This drastically reduces decoding latency and allows μSDR to quickly respond to incoming packets. In μSDR, the ACK frame generation is also implemented in hardware. The ACK generation block pre-stores preambles and SFD in a small 11-byte ACK FIFO. The only missing parts of the ACK are the data sequence number (DSN) and corresponding FCS. If the incoming packet passes the CRC check, the hardware copies the packet's DSN into the ACK FIFO. At the same time, the ACK generator block calculates the corresponding FCS and initiates a countdown for precise timing. In our IEEE 802.15.4 implementation, the receiver calculates the CRC and FCS within 12 μs, leaving 180 μs of slack to meet the timing requirements.

4.5 Operating System

In our μSDR implementation, the transmission and receiving chain are synthesised in the FPGA to reduce the workload of the MCU. However, given the high speed interconnect between the MCU and the FPGA, the computation boundary could be flexible depending on the radio's architecture. We measured the latency of software-implemented symbol decoding. Each symbol consists of 32 chip sequences and occupies 16 μs in time. Symbol decoding computes the Hamming Weight of a given 32-bit sequence and XORs it to 16 known 32-bit sequences. Each symbol can be decoded in 1,679 cycles, which is 16.8 μs at 100 MHz. 16.8 μs is on the edge of real-time decoding. We believe it may be possible to implement most of the receiving chain in software and maintain the timing constraint with more optimized code. The transmission chain consists of several LUTs and requires less computation than the receiving chain. In μSDR implementation, the data rate from the FPGA fabric to DAC is set to 48 MBps, much less than the AHB bandwidth between the MCU and FPGA. Therefore, transmission can be implemented entirely in software with ease.

4.6 Radio Enhancements

The receiver, transmitter, and hardware acknowledgment implementations are everything needed to fully interact with commercially available IEEE 802.15.4 radios. The following sections describe enhancements to support more advanced MAC features that are not always found in currently available radios. This highlights the strength of μSDR to test and validate new radio architectures that jointly optimize low-power MAC protocols.

4.6.1 Multiple Address Recognition

The most important decision a low-power MAC protocol makes is to stay awake or go back to sleep if channel activity has been detected. Recent advances in low-power MAC protocols showed that receiver initiated protocols have an advantage over low-power listening types of protocols as they are more robust against interfering noise [6]. In a receiver initiated MAC protocol, nodes that want to receive packets transmit a short probe. Nodes that have packets to transmit buffer them locally until they hear a beacon from the intended receiver. Once they decoded a beacon, the nodes acknowledge it and start transmitting the data in the next packet. A-MAC demonstrated that this could be done in a very efficient way, even if multiple nodes try to send data to the same receiver [6]. A-MAC exploits the fact that acknowledgment messages will interfere constructively at the receiver. This allows the receiver to make the crucial decision to stay awake, even if the first data packet of the multiple transmitters collides, at which point a back-off mechanism must resolve the conflict.

Prior work identified one critical factor limiting the performance of A-MAC is the radio address recognition mechanism [6]. Hardware acknowledgments can only be generated for ones own address. Thus, A-MAC reprogrammed it's own address to the intended receiver's beacon address (receiver's address with the MSB bit set) waiting for the beacon. This introduced several problems which limit the protocol, chief among them the ability to listen for only one node at a time.

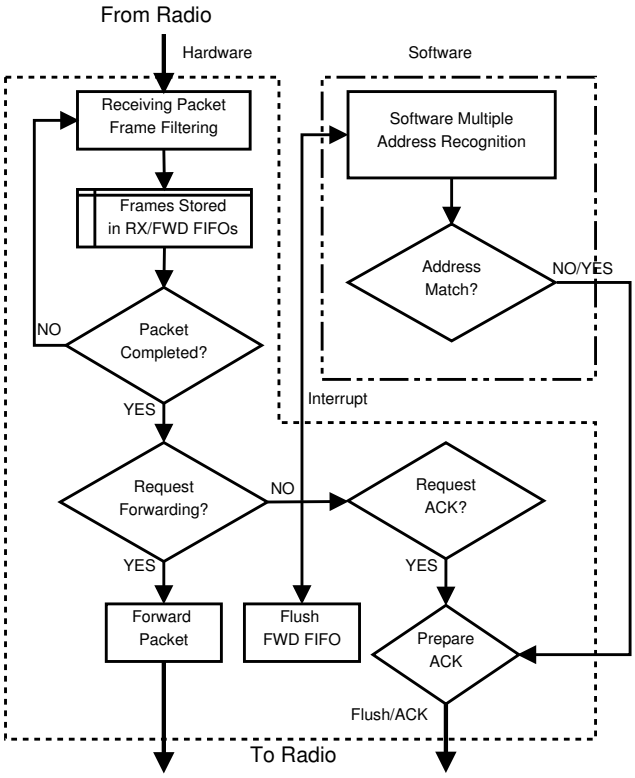


Figure 6: Packet reception and response in μSDR. This diagram shows the partitioning of responsibilities between the hardware and software portions of μSDR. The decision to support multiple address recognition requires software support of the reception/response loop to keep hardware area costs reasonable. In this tightly-coupled model, the software controller takes advantage of the hardware accelerated acknowledgments to meet the stringent latency requirements of the 802.15.4 acknowledgment frame.

We implemented a multiple address recognition mechanism in μSDR. According to the IEEE 802.15.4 standard, the address field can be as long as 8 bytes. This leads to a design trade-off between implementing this functionality in software or hardware by trading latency for FPGA area. We decided to have a slight latency increase, and implemented a multiple address recognition algorithm on the processor that can detect up to 8 addresses. However, this latency increase does not impact the performance of the protocol, as μSDR can read out the address faster from the receive FIFO than the time it has to wait to reply with an ACK. In addition, the multiple address recognition algorithm takes advantage of the hardware acknowledgment mechanism preparing an ACK while receiving the incoming packet. All the algorithm has to do is send a signal to the peripheral letting it know to acknowledge this message. Figure 6 shows the state diagram of the algorithm and how hardware and software interact. This demonstrates the flexibility of the platform, allowing certain protocol functionalities that are time critical to be done in hardware only (e.g., hardware acknowledgments generation) while others are implemented in software.

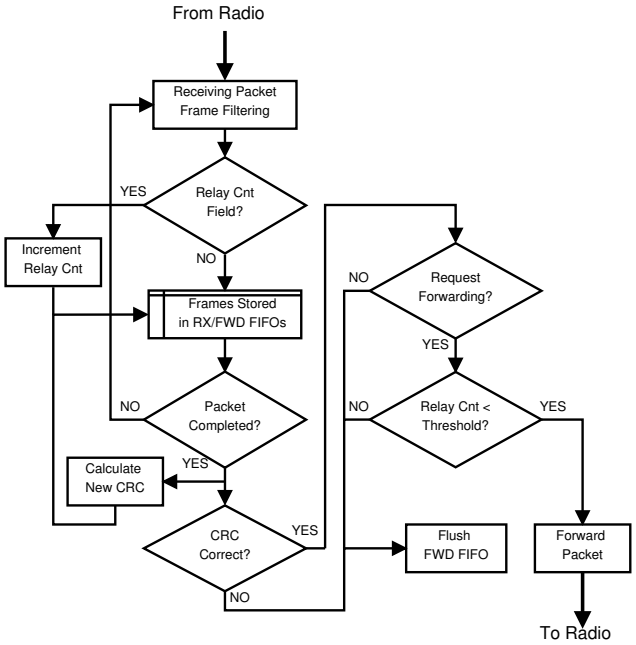


Figure 7: Detailed packet forwarding state diagram. This is a more detailed version of the packet forwarding mechanism shown on Figure 6. Packets are stored in an incoming FIFO, automatically incrementing the Relay Count field and CRC as necessary. When a packet completes, it is automatically forwarded if requested and its count is below the relay threshold.

4.6.2 Packet Forwarding

Two basic services in wireless sensor networks are time synchronization and flooding. The broadcast storm is a common problem flooding algorithms have to take care of [20], as retransmission of regular broadcasts can amplify itself, and thus overload the network. Ferrari et al. proposed *Glossy* [11], a fast flooding and time synchronization mechanism. Glossy exploits the fact that in IEEE 802.15.4, if multiple identical packets arrive within 0.5 μs at a receiver, they collide constructively, and the receiver can successfully decode the message with high probability. To achieve this, Glossy nodes forward a received packet during a precise time slot and increment a relay counter in the packet. Packets with identical relay counts will be transmitted at the same time, at all nodes. Therefore, the packets will constructively interfere at the receivers. The relay counter limits the maximum hop count of the packet, making sure that packets do not live forever within the network. Glossy uses a fragile software technique to guaranty software latencies. They have to disable all interrupts and add no-operation (NOP) commands to adjust the latencies during a Glossy transfer. Although the wireless links are lossy, Glossy demonstrates 99.9999% flooding reliability in a 10 hop deep network.

Österlind et al. proposed a zero-copy mechanism [24] to improve the throughput of multi-hop wireless sensor networks. He observes that the critical path of zero-copy is the interface between radio and processor. The packet forwarding is done by downloading the packet from the radio to the processor, and uploading it back into the radio. Instead of downloading and then uploading the same packet, it would be advantageous if the radio stored the received packet and were able to immediately retransmit it out. This method improves the total throughput to 97% of the theoretical upper bound.

Voltage Rail	SmartFusion Static Current	Spartan Static Current
3.3 V (I/Os)	26.9 mA	79 mA
2.5 V (Aux)	n/a	88 mA
1.5 V (Core)	8.0 mA	n/a
1.2 V (Core)	n/a	80 mA
Total Power	101 mW	578 mW

Table 4: Static current at each supply rail for the SmartFusion and Spartan 3-2000. The Spartan is a more traditional SRAM-based FPGA whereas the SmartFusion is a flash-based design. Measurements are taken after both devices have been configured and allowed to settle. We observe less static current at the I/O buffers, and significant differences at the core supply. The SmartFusion current is $10\times$ less than the Spartan. As a consequence, the SmartFusion requires approximately 18% as much static power as the Spartan, making flash-based FPGAs a much better design decision for low-power platforms.

We decided to implement a fast packet forwarding mechanism in the FPGA of the μSDR platform. We use the data sequence number (DSN) as a relay counter. Figure 7 shows the state diagram. During reception of a packet, the receiver stores the data in two different FIFOs, the receiver and forwarding FIFOs. However, the two FIFOs store slightly different information in the DSN and frame control sequence (FCS). The DSN gets automatically incremented by 1, and the FCS is updated accordingly.

Using this mechanism, μSDR forwards a packet without interacting with the software at all. This removes the critical path identified by Österlind and allows the retransmit to happen in less than 20 μs. This forwarding also solves the fragile Glossy software implementation as no interrupts have to be disabled on the software side, nor do we have to add any NOPs. In addition, the forwarding can be precisely controlled, changing the latency to any arbitrary, but fixed, delay (e.g., to 192 μs which would correspond to the ACK turn around time). This would allow backward compatibility with some existing commercial radios that have forwarding capabilities (e.g., the TI CC2520), but have a transmit to receive turn around time of 192 μs as specified by the IEEE 802.15.4 standard.

5 Evaluation

We evaluate μSDR on its merits as a low-power SDR platform. Specifically, we focus on its power draw, latency, and flexibility. We further examine the ability and performance of the μSDR system in supporting low-power IEEE 802.15.4 physical and link layer protocols.

Frequency Scaling	Wake-up Latency	System Current	System Power
off	34 μs	108 mA	518 mW
on	3.01 ms	67 mA	322 mW

Table 5: Sleep power and latency trade-off. Without frequency scaling, μSDR draws 108 mA while Waiting For an Interrupt (WFI). It wakes-up from this state in only 34 μs. With frequency scaling, μSDR saves 38% in power while taking 89$\times$ longer to wake up.

5.1 Platform Micro-Benchmarks

Designing a low-power system requires building from the ground up with low-power components. In traditional software radio designs, one of the largest power sinks is the FPGA. The relatively recent emergence of flash-based FPGAs augmented with a low-power microcontroller opens the door to low-power design. In Table 4 we compare the static power draw of the flash-based SmartFusion to the SRAM-based Xilinx Spartan 3-2000 used in the USRP 2. The superior FPGA core power efficiency of the flash-based FPGA is a direct consequence of the technology. Freed from the comparatively high leakage currents of SRAM logic cells, the flash-based FPGA draws very little static power. As a young technology, we expect these innovations in static power for flash-based FPGAs to improve rapidly, as evidenced by the Microsemi IGLOO series, whose Flash*Freeze technology lowers the FPGA static power draw down to just 2 μW [16].

In addition to the impact on static power draw, flash-based FPGAs also improve cold-boot performance. For an FPGA to be useful, it must at some point be configured. In the case of SRAM-based FPGAs, this configuration must occur at every boot, as the configuration of the FPGA is lost when power is no longer supplied. This configuration period introduces a non-trivial delay for the FPGA wake-up process as well as consuming a relatively large amount of energy. Most importantly, the latency imposed by the configuration window inhibits effective duty-cycling of the FPGA. In Figure 9, we compare the cold-boot times of the SRAM-based USRP 2 (9(a)) to the μSDR (9(b)). After a system in-rush current spike of 1.8 A, the USRP 2 averages a 700 mA current draw for 1.964 s while it configures its FPGA. This long configuration process and high current draw consume 9.56 J of energy, a cost that must be paid every time the FPGA is power cycled.

In contrast, μSDR has no current spike on bootup and offers an average current draw of under 120 mA. To measure time to first useful instruction, we wrote a test program that simply toggles an I/O line, indicating the processor is ready to start executing. We measured the time from power application until the I/O asserts. The μSDR requires 54 ms to boot the system, consuming only 0.028 J. Even the latency of μSDR is not ideal for a rapidly duty-cycled system, which should be measured in *microseconds* rather than in *milliseconds*. μSDR is still significantly faster than the USRP 2's *seconds* to boot: μSDR offers 36$\times$ lower cold-boot latency and 341$\times$ lower energy usage.

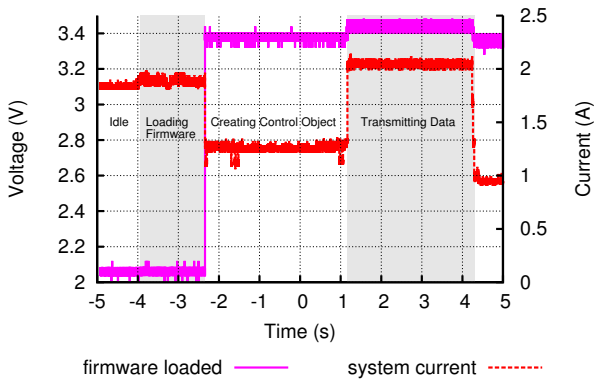
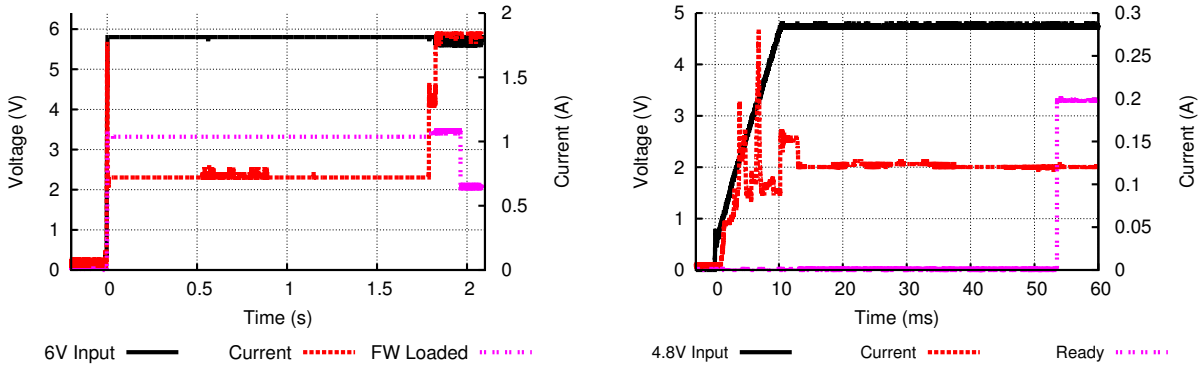


Figure 8: USRP E100 power profile. The E100 runs a full embedded Linux platform, making cold-boot an untenable comparison. Rather we show the timing from an idle, booted system to load a firmware image (configure the FPGA) and send the data. The firmware loaded signal is recorded from a firmware loaded pin exported by the E100. After configuration, the system idle power drops to just under 1 A at 6 V.

Given that the 54 ms boot time is still too slow to support fine-grained duty-cycling, μSDR also presents a sleep mode as a median option between system full on and full off. Unfortunately, the SmartFusion chip is only capable of sleeping the M3 core (via WFI[2]). Fully sleeping the FPGA would require technology similar to the Flash*Freeze mode from the Microsemi IGLOO line of low-power FPGAs. Unfortunately these FPGAs do not yet support the tight integration with a hard CPU and are currently unsuitable for the μSDR platform. Until technology similar to Flash*Freeze comes to the SmartFusion, μSDR is still capable of lowering the FPGA power via frequency scaling. In μSDR's frequency scaling mode we turn off the external crystal oscillator and used the FPGA's internal RC network instead. The frequency of the RC network is reduced to 6.5% of the original frequency (to 3.125 MHz). We additionally turn off the PLL and bypass two other clock sources for FPGA logic to further reduce the system current. Table 5 details the results of these power optimizations. Placing the processor in WFI state and power gating all external components reduces system current draw to 108 mA with only 34 μs of wakeup latency. Introducing FPGA frequency scaling reduces system power further to 67 mA, but imposes a 3.01 ms latency to start the crystal oscillator and stabilize the PLL before the processor can begin execution.

Finally, we compare μSDR to the USRP E100, USRP's "embedded" series. The E100 is built on the GumStix platform, a small ARM-based PC-like device. The advantage of the E100 is it requires no controlling computer; rather the controller is built in to the system. As a consequence, the E100 actually has greater power draw than its PC-encumbered counterpart, as Figure 8 shows.

[2]WFI: Wait For Interrupt, an ARM instruction that places the core in a lower power sleep state until interrupted. The contents of registers are preserved but the rest of the core is power gated



(a) USRP 2 Cold-Boot

(b) μSDR Cold-Boot

Figure 9: USRP 2 and μSDR cold-boot comparison. Both systems power on at 0 s. As expected, the USRP 2 (a) has a current spike of 1.8 A, and exhibits an average current draw of 700 mA at 6 V. It takes 1.96 s to load the firmware from an external SD card, resulting in 9.56 Joules of energy consumption during cold-boot. In contrast, the flash-based μSDR does not have an in-rush current and the average current is less than 120 mA at 4.8 V. Moreover, μSDR requires 54 ms to boot the system, resulting in only 0.028 Joules of energy consumption. μSDR is 36$\times$ faster and uses 341$\times$ less energy than the USRP 2 to start-up.

5.2 Application Micro-Benchmarks

We evaluate how well A-MAC and Glossy, two recently proposed low-power protocols, are supported by μSDR.

5.2.1 *Validating A-MAC Acknowledgments*

Prior work proposed using ACK frames to support a receiver-initiated MAC protocol [6]. Unfortunately that work was built upon the fixed function CC2420 radio, which does not allow sufficient flexibility to fully realize the protocol. We investigate the A-MAC protocol's colliding ACK primitive on the μSDR platform, both to validate the original A-MAC protocol and to demonstrate the capabilities of the μSDR platform.

In A-MAC we can assume two nodes transmit the same packet $d(t)$ with carrier frequencies f_1 and $f_1+\Delta f$ at the same time. The transmitted RF-signal $s(t)$ then is achieved by summing the $d(t)$ modulated with each carrier frequency f_1 and $f_1+\Delta f$:

$$s(t) = d(t) \times [cos(2\pi \times f_1 \times t) + cos(2\pi \times (f_1+\Delta f) \times t)]$$

This equation can be simplified using the sum-to-product identity:

$$s(t) = d(t) \times \left[2 \cdot cos(2\pi \times \tfrac{2f_1+\Delta f}{2} \times t) \times cos(2\pi \times \tfrac{\Delta f}{2} \times t)\right]$$

Assuming that the receiver has the same carrier frequency f_1, the received RF-signal $r(t)$ can be simplified as:

$$r(t) = s(t) \times cos(2\pi \times f_1 \times t)$$

By down conversion, the received baseband signal $r_B(t)$ is equivalent to $r(t)$ without high frequency component, which can be expressed as:

$$r_B(t) = d(t) \times cos^2(2\pi \times \tfrac{\Delta f}{2} \times t)$$

The takeaway here is the existence of an envelope frequency of $\Delta f/2$ that encompasses the received ACKs.

Figure 10 shows an example of a packet collision and the resulting envelope. The envelope introduces *local minima* to the received signal. These are important as the signal amplitude is attenuated during a local minimum, which may cause the receiver to incorrectly decode the signal. Local minima in the received signal occur whenever the following condition holds:

$$2\pi \times \tfrac{\Delta f}{2} \times t = \tfrac{\pi}{2} \times n \quad n \in N$$

Prior work measured the acknowledgment reception rate (ARR) for varying numbers of concurrently transmitting neighbors, and found that in one experiment, the worst case ARR was 97% [8].

As an example of a radio optimization enabled by μSDR but unavailable to high-latency SDRs or fixed-function radios, we explore the effect of automatic gain control (AGC) methods on packet reception rate from two concurrent transmitters. The AGC algorithm detects and compensates for varying strength signals by dynamically adapting the amplitude of the raw RF signal for further processing. The available AGC resolution and responsiveness depend on the latency of the AGC controller. Highly latent devices such as the USRP platform can only do AGC on a *per-packet* basis whereas μSDR is capable of doing AGC on a *fixed-latency* basis. In addition to latency issues, many SDR platforms do not provide sufficient introspection into the RF frontend to allow for fine-grained AGC, relying instead on metrics such as RSSI.

Traditional AGC in commodity radios latch a gain value upon receiving the Start of Frame Delimiter (SFD). This AGC loop is sufficient if the amplitude of a signal is constant over the entire packet. However, if multiple nodes transmit concurrently, a radio with an SFD-latched AGC may experience a changing signal amplitude over time from the envelope modulation as seen in Figure 10. We implemented both an SFD-latched AGC and a continuous AGC in μSDR.

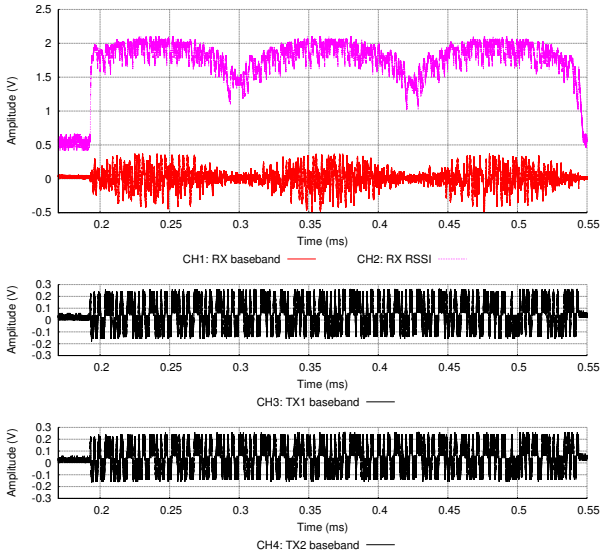


Figure 10: A constructive ACK collision is observed. CH1 is the RX baseband signal. CH2 is the RSSI. CH3 and CH4 are the TX baseband signals of the two colliding ACKs. The slightly offset carrier frequencies of CH3 and CH4 interact to form the envelope modulation on the received baseband signal. Without automatic gain control, varying signal strength resulting from the envelope severely hinders the receiver's ability to successfully decode the signal.

Table 6 summarizes the results of a basic comparison. The results show that continuous AGC offers a small improvement in the ARR over SFD-latched AGC.

Further experimentation with AGC revealed that the properties of each AGC method rely heavily on the degree of separation between the two carrier waves. Figure 11 plots the reception rate of concurrently transmitted packets against varying differences in the two transmitting carrier wave frequencies. Recall that the period of the beat frequency of the modulating envelope is given by $T = \frac{1}{\Delta f}$. For lower values of Δf, this means the period of envelope beats will be relatively long and the continuous AGC is able to adapt and correct for the variation in signal strength. As Δf increases, however, the local minima from the envelope wave increase in frequency (while decreasing in length). Eventually the continuous AGC cannot adapt fast enough to the varying signal strength imposed by the envelop.

AGC mode	Average ARR	Std
SFD-latch	97.27%	0.211%
Continuous	98.20%	0.065%

Table 6: Acknowledgment reception rate (ARR) for two constructively interfering transmitters with respect to different AGC modes. We transmitted 10,000 ACKs per transmitter per experiment and repeated each experiment 5 times. A continuous AGC performs slightly better due to the small carrier frequency separation of the transmitting nodes.

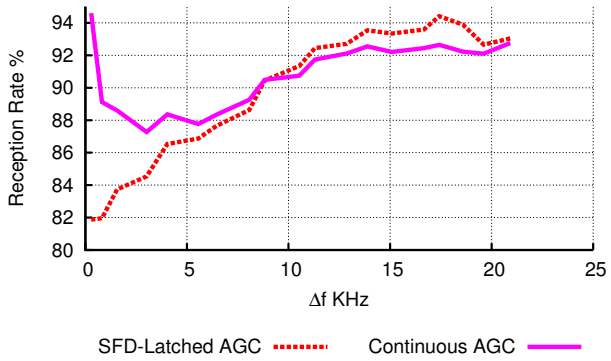


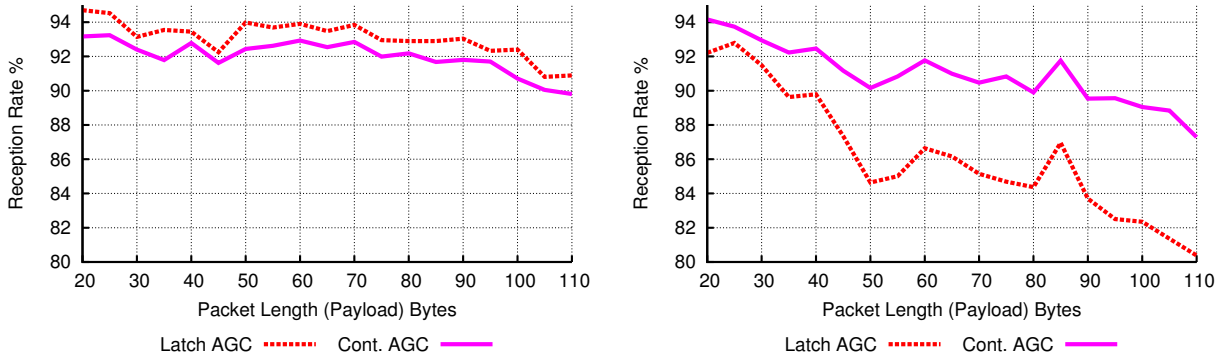
Figure 11: Reception rate versus carrier frequency separation of two concurrent transmitters with a fixed packet length (60 bytes). The period of the beat frequency of the enveloping modulation is $T = \frac{1}{\Delta f}$. For small Δf, this period is sufficiently long that continuous AGC is able to correct for the varying signal strength (compare Figure 10 CH3). As Δf grows, the beat period shortens until it is too fast for continuous AGC to keep up. At this point, however, the minima are sufficiently narrow to only obscure a few chips and the spreading built into 802.15.4 recovers the missing information. The continuous AGC's attempt to follow the high-frequency minima account for for the slightly worse performance of continuous AGC at higher Δfs. This illustrates how low-level control can improve protocol performance.

Once the AGC loop latency is greater than the beat period, the minima are so short that only a few *chips* are lost. The 802.15.4 protocol employs a spreading technique such that 4-bit of data are composed into 1 symbol made up of 32 *chips*. The redundancy supplied by the spreading means that once the beat frequency of the envelope is too high to correct via AGC, only a few *chips* of the symbol are dropped, allowing the symbol as a whole to be correctly decoded. This phenomena explains the observed upward trend of the SFD-Latch AGC as Δf increases.

As Δf grows sufficiently large, the frequency of the minima begins to approach and ultimately surpass the speed of the continuous AGC control loop. The loop delay of the continuous AGC calculation then causes the actual gain to be applied to the incoming signal too late. The extra strength oscillation imposed by the late gain adaptation accounts for the 1~2% worse performance of continuous AGC versus the latched AGC for high Δfs.

5.2.2 *Validating Glossy Broadcast Collisions*

Thus far we have focused exclusively on ACK collisions, which are relatively short packets (11 bytes). In Glossy, the authors observed that the same constructive ACK-collision optimization could be applied to broadcast packets. We implemented a naïve broadcast, each node flooding the network. In this broadcast scheme, the packet forwarding time is controlled precisely which allows multiple forwarded packets to (ideally) interfere constructively. Unlike the short, fixed-length ACK packets, the forwarded broadcast packets may be arbitrarily long.



(a) RF frequencies separate by 16.5 KHz

(b) RF frequencies separate <1 KHz

Figure 12: Reception rate of constructively interfering packet collisions with the transmitter at two slightly different carrier frequencies. Both transmitters send the exact same message, at the same time. Figure (a) shows the result when the transmitters are separated by 16.5 KHz, while Figure (b) depicts the case of the transmitters separated by <1 KHz. In both cases, the longer the packets, the lower the reception rate as we get more beats in a single packet. This leads to a lower signal amplitude, and thus potential for decoding errors. To mitigate this, the AGC can be held constant after latching it at the SFD detection, or it can continuously updated over the whole packet length. For small carrier frequency offsets (Figure (b)) continuously updating the AGC improves the reception rate by 2∼6%.

We can infer from the previous work that *reception rate should drop if more envelopes exist during a given packet*. Given two nodes with a fixed but slightly different carrier frequencies, we measured the reception rate across different packet lengths. Figure 12 shows the expected result: as the packet length increases, the reception rate decreases. The reception rate is derived from the number of packets with a successfully decoded CRC value over 30,000 transmissions. If any symbol in the packet is incorrectly decoded, that packet will fail the CRC check.

5.3 Frequency Compensation

Assuming the transmitter T and receiver R has corresponding carrier frequency f_{c1}, f_{c2} respectively, and $f_{c1} \neq f_{c2}$. The frequency mismatch affects the received in-phase (I) and quadrature-phase (Q) signal. Assuming the transmitted in-phase and quadrature-phase baseband signal are $S_I(t)$, $S_Q(t)$, the actual signal which is being transmitted through the air $S(t)$ can be written as follows:

$$S(t) = S_I(t) \cdot cos(2\pi \cdot f_{c1} \cdot t) + S_Q(t) \cdot sin(2\pi \cdot f_{c1} \cdot t)$$

Ignoring the channel characteristics and assuming no noise or interference, the received signal $R(t) = S(t)$. The received in-phase signal can be expressed as follows.

$$\begin{aligned} r_I(t) =& R(t) \cdot cos(2\pi \cdot f_{c2} \cdot t) \\ =& [S_I(t) \cdot cos(\omega_{c1} t) + S_Q(t) \cdot sin(\omega_{c1} t)] \cdot cos(\omega_{c2} t) \\ =& S_I(t) \cdot cos(\omega_{c1} t) \cdot cos(\omega_{c2} t) + S_Q(t) \cdot sin(\omega_{c1} t) \cdot cos(\omega_{c2} t) \end{aligned}$$

Where $\omega_{c1,2} = 2\pi \cdot f_{c1,2}$. Simplifying the equation by the product-to-sum identities.

$$\begin{aligned} r_I(t) =& \tfrac{1}{2} S_I(t) [cos((\omega_{c1} - \omega_{c2})t) + cos((\omega_{c1} + \omega_{c2})t)] + \\ & \tfrac{1}{2} S_Q(t) [sin((\omega_{c1} - \omega_{c2})t) + sin((\omega_{c1} + \omega_{c2})t)] \end{aligned}$$

By removing high frequency components $(\omega_{c1} + \omega_{c2})$, the received in-phase/quadrature-phase baseband signal $r_{IB}(t)$, $r_{QB}(t)$ can be expressed as following.

$$\begin{aligned} r_{IB}(t) &= \tfrac{1}{2} S_I(t) \cdot cos(\Delta\omega t) - \tfrac{1}{2} S_Q(t) \cdot sin(\Delta\omega t) \\ r_{QB}(t) &= \tfrac{1}{2} S_I(t) \cdot sin(\Delta\omega t) + \tfrac{1}{2} S_Q(t) \cdot cos(\Delta\omega t) \end{aligned}$$

where $\Delta\omega = \omega_{c2} - \omega_{c1}$. By rearranging these equations, we can express the following.

$$\begin{bmatrix} r_{IB}(t) \\ r_{QB}(t) \end{bmatrix} = \begin{bmatrix} cos(\Delta\omega t) & -sin(\Delta\omega t) \\ sin(\Delta\omega t) & cos(\Delta\omega t) \end{bmatrix} \times \begin{bmatrix} 1/2 \cdot S_I(t) \\ 1/2 \cdot S_Q(t) \end{bmatrix}$$

Thus the frequency mismatch is equivalent to the rotation of the complex coordinate with angular velocity $\Delta\omega$. Hence, by measuring the rotation speed and direction, we can back calculate the frequency offset between receiver and transmitter. In the O-QPSK modulation, the I/Q signal at any given time map to a unit circle on a constellation. Measuring both clockwise and counter-clockwise rotation angle at fixed intervals allows the system to determine whether the receiver's carrier frequency is leading or lagging. With this information, μSDR is able to compensate for the frequency mismatch in a subsequent transmission, so we call it automatic frequency compensation (AFC).

Figure 13 shows the simulated result of running AFC on the μSDR platform. In this simulation, f_{c1} and f_{c2} are set 50 KHz apart and the step size of f_{c2} is 300 Hz. After receiving a packet, the AFC adapts the receiver carrier frequency f_{c2} to reduce the frequency mismatch. Two factors are important for AFC effectiveness: packets with higher SNR afford more accurate angle estimation and packets with longer payloads provide more opportunities to adapt the carrier frequency. Figure 13 shows how variations in these properties affect reception with AFC.

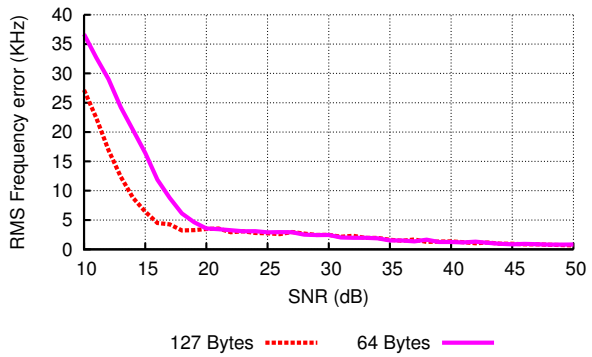


Figure 13: AFC simulation. Frequency separation of TX and RX is 50 KHz initially ($f_{c1} - f_{c2} = 50\ KHz$). The figure shows the RMS frequency separation after receiving a packet with AFC enabled. As expected, AFC performance improves with higher SNR and greater packet length.

In Table 7, we explore the combination of AFC with the previously explored AGC. Combining the continuous AGC with AFC yields the best result, a 95.5% ARR. Applying AFC to the simpler SFD-latched gain control results in a 1.2% reduction in ARR, however. Recall from the AGC discussion and Figure 11 that SFD-latched gain control performs poorly for small Δf as the local minima from the envelope beat are long. As the AFC attempts to compensate for the frequency difference between TX and RX, these minima become longer which improves the performance of the adaptive AGC, but results in more below-threshold signal windows for the fixed-gain, SFD-latched method.

6 Discussion

We built μSDR with the intention to see how close we can get to the claims presented by Dutta et al. [7]. This section will look at the four different requirements *radio duty-cycling*, *low-power FPGA*, *system integration*, and *measurement* and compare the current μSDR platform to the performance it is supposed to achieve to compete with current commercially available low-power wireless radios.

AGC mode	AFC mode	ARR
SFD-latch	Enable	93.3%
Continuous	Enable	95.5%
SFD-latch	Disable	94.5%
Continuous	Disable	95.1%

Table 7: Acknowledgment reception rate (ARR) for two constructively interfering transmitters with respect to AFC and different AGC modes. We transmitted 10,000 ACKs per transmitter per experiment. With AFC disabled, carrier frequencies of TX and RX were off by 16.4 KHz. In both cases, continuous AGC worked better. Enabling AFC worsened the reception rate for SFD-Latch AGC because the baseband signal is attenuated by the low-frequency envelope.

6.1 Radio Duty-Cycling

Typical radio duty-cycling platforms achieve <1 mW sleep power, wake up in μs, and draw tens to hundreds of mW while actively using the radio. μSDR draws in its lowest sleep state 322 mW (wakeup in 3.01 ms) or 518 mW (wakeup in 34 μs). During transmit, μSDR draws 1.4 W, and 1 W while receiving. While the sleep current is still three orders of magnitude greater than typical duty-cycled systems, it offers a starting point to explore the space of battery-powered SDR platforms. Comparing this to an iPhone 4S with a battery capacity of 5.3 Wh, we can power μSDR for 3.8 hours while constantly transmitting, or 16.4 hours in deep sleep. Thus, using a moderate amount of duty-cycling, a 12 hour deployment time could be achieved easily.

6.2 Low-Power FPGA

So where does all the power go during sleep? All radio frontend components, ADC, DAC, and the baseband converter support sleep modes in the μW range. Still, μSDR draws hundreds of mW. Unfortunately, the SmartFusion does not include the latest Flash*Freeze technology which would allow it to clock-gate the FPGA itself. Thus, even in deep sleep, the FPGA draws considerable dynamic power. We expect that future versions of the SmartFusion will include such technology, as it already exists in the IGLOO flash-based FPGA family offered by the same vendor.

6.3 System Integration

μSDR significantly departed from the common platform reconfigurability of typical SDR systems, integrating everything onto a 102 cm^2 sized PCB. This reduced cost to <$150. While we did not achieve the expected $100 suggested by Dutta et al. [7], we are within striking distance. With larger quantities and the commoditization of these mixed-signal FPGAs with hard-silicon cores, the total price will and drop below the target $100 mark.

6.4 Measurement

The current iteration of the μSDR platform does not include any energy metering capabilities that are exposed to the application processor. While we can measure the external power draw on the different power rails, there are no components of the SmartFusion connected to meter those rails internally. However, the SmartFusion has a full Analog Compute Engine with up to 12 direct ADC inputs. The current μSDR platform does not take advantage of any of them, and we plan to add this support in the next iteration of the platform.

7 Conclusions

Software-defined radios are reconfigurable communication systems that transcend historical boundaries between hardware and software subsystems, physical and logical layers, and analog and digital domains. In so doing, they enable radical new architectures, novel radio designs, and high-performance protocols that are not easy to design, implement, or evaluate using traditionally-layered approaches. Although modern SDR platforms have been used to explore many facets of the wireless design space, their current architectures make it very difficult to explore the *low-power* design space. Their use of SRAM-based FPGAs result in high static and dynamic power draws, their slow startup times are not amenable to rapid duty cycling, their radio frontends

do not support power controls, and their processing requirements place a heavy load on the system. As a result, fertile application areas like mobile phones and sensor networks that could benefit from radical approaches, but which require low-power operation, remain relatively unexplored.

We developed μSDR to address this inequity. This paper demonstrates that a software radio with a footprint of 100 cm^2 and cost of $150 is able to operate from a pack of 'AA' batteries, and that we can expect a lifetime similar to today's smart phones. This work enables new research areas that were completely out-of-reach or existed only in severely limited forms in low-power nodes. Hence, μSDR is an enabling technology for many high-impact, large scale applications of low-power, ad-hoc wireless networking where high performance or precise timing are required, including full-duplex wireless communication, synchronous concurrent communication, high-frequency power metering, infrastructure less audio/video streaming, and structural health monitoring.

8 Acknowledgments

We thank Vijay Raghunathan, our shepherd, and the anonymous reviewers for their valuable feedback. This material is based upon work supported by gifts from Microsemi Corporation and by the National Science Foundation under Grant #0964120 ("CNS-NeTS"), #1059372 ("CI-ADDO-NEW"), and #1111541 ("CNS-CSR"). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

9 References

[1] Joint Tactical Radio System. http://jpeojtrs.mil.

[2] Altera. Cyclone V SoC FPGA Hard Processor System. http://www.altera.com/devices/fpga/cyclone-v-fpgas/hard-processor-system/cyv-soc-hps.html.

[3] K. Amiri, Y. Sun, P. Murphy, C. Hunter, J. Cavallaro, and A. Sabharwal. Warp, a unified wireless network testbed for education and research. In *International Conference on Microelectronic Systems Education 2007. (MSE '07)*.

[4] Analog Device. AD9288: 8-Bit, 40/80/100 MSPS Dual A/D Converter. http://www.analog.com/static/imported-files/data_sheets/AD9288.pdf.

[5] P. Dutta, D. Culler, and S. Shenker. Procrastination might lead to a longer and more useful life. In *Proceedings of the 6th ACM SIGCOMM Workshop on Hot Topics in Networks 2007*, (Hotnets '07).

[6] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. M. Liang, and A. Terzis. Design and evaluation of a versatile and efficient receiver-initiated link layer for low power wireless. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems 2010*, (SenSys '10).

[7] P. Dutta, Y.-S. Kuo, A. Ledeczi, T. Schmid, and P. Volgyesi. Putting the software radio on a low-calorie diet. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks 2010*, (Hotnets '10).

[8] P. Dutta, R. Musăloiu-E, I. Stoica, and A. Terzis. Wireless ack collisions not considered harmful. In *Proceedings of the 7th ACM SIGCOMM Workshop on Hot Topics in Networks 2008*, (Hotnets '08).

[9] Ettus Research. USRP E100. https://www.ettus.com/product/details/UE100-KIT.

[10] Ettus Research. USRP N200. https://www.ettus.com/product/details/UN200-KIT.

[11] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks 2011*, (IPSN '11).

[12] Intel. Intel pentium mobile processor, Apr. 2012.

[13] JTRS Standards. Software Communications Architecture Specification. http://jpeojtrs.mil/sca/Documents/SCAv4_0/SCA_4.0_20120228_ScaSpecification.pdf.

[14] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner. Soda: A low-power architecture for software radio. In *Proceedings of the 33rd International Symposium on Computer Architecture 2006*, (ISCA '06).

[15] Maxim. MAX5189: Dual, 8-Bit, 40MHz, Current/Voltage, Simultaneous-Output DACs. http://datasheets.maxim-ic.com/en/ds/MAX5186-MAX5189.pdf.

[16] Microsemi. IGLOO Series. http://www.actel.com/products/iglooseries/default.aspx.

[17] Microsemi. SmartFusion: Customizable System-on-Chip (cSoC). http://www.actel.com/documents/SmartFusion_DS.pdf.

[18] G. Minden, J. Evans, L. Searl, D. DePardo, V. Petty, R. Rajbanshi, T. Newman, Q. Chen, F. Weidling, J. Guffey, D. Datla, B. Barker, M. Peck, B. Cordill, A. Wyglinski, and A. Agah. Kuar: A flexible software-defined radio development platform. In *2nd IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks 2007*, (DySPAN '07).

[19] J. Mitola III. SDR architecture refinement for JTRS. In *Proceedings of 21st Century Military Communications Conference 2000*, (MILCOM '00).

[20] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the 5th ACM/IEEE Annual International Conference on Mobile Computing and Networking 1999*, (MobiCom '99).

[21] J. Notor, A. Caviglia, and G. Levy. Cmos rfic architectures for ieee 802.15.4 networks. Cadence Design Systems, Inc.

[22] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste. Enabling mac protocol implementations on software-defined radios. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation 2009*, (NSDI '09).

[23] Octasic Inc. Vocallo BTS.

[24] F. Österlind and A. Dunkels. Approaching the maximum 802.15.4 multi-hop throughput. In *Proceedings of the 5th ACM Workship on Embedded Networked Sensors 2008*, (HotEmNets '08).

[25] C. Patridge. Realizing the future of wireless data communications. *COMMUNICATIONS OF THE ACM*, 54(9):62–68, September 2011.

[26] K. Piotrowski, P. Langendoerfer, and S. Peter. How public key cryptography influences wireless sensor node lifetime. In *Proceedings of the 4th ACM Workshop on Security of Ad Hoc and Sensor Networks 2006*, (SASN '06).

[27] T. Schmid. GNU Radio 802.15.4 En- and Decoding. http://nesl.ee.ucla.edu/fw/thomas/thomas_project_report.pdf.

[28] T. Schmid, O. Sekkat, and M. B. Srivastava. An experimental study of network performance impact of increased latency in software-defined radios. In *Proceedings of the 2nd ACM Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization 2007*, (WinTECH '07).

[29] R. Szewczyk, A. Mainwaring, J. Polastre, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems 2004*, (SenSys '04).

[30] K. Szlavecz, A. Terzis, S. Ozer, R. Musaloiu, J. Cogan, S. Small, R. Burns, J. Gray, and A. Szalay. Life under your feet: An end-to-end soil ecology sensor network, database, web server, and analysis service. Technical Report Microsoft Technical Report MSR TR 2006 90, Microsoft Research.

[31] K. Tan, J. Zhang, J. Fang, H. Liu, Y. Ye, S. Wang, Y. Zhang, H. Wu, W. Wang, and G. Voelker. Sora: high-performance software radio using general-purpose multi-core processors. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation 2009*, (NSDI '09).

[32] Texas Instruments. ADC081S101: Single Channel, 8-Bit ADC. http://www.ti.com/lit/ds/symlink/adc081s101.pdf.

[33] Xilinx. ZYNQ-7000: All Programmable SoC. http://www.xilinx.com/publications/prod_mktg/zynq7000/Product-Brief.pdf.