WiBench: An Open Source Kernel Suite for Benchmarking Wireless Systems

Qi Zheng*, Yajing Chen*, Ronald Dreslinski*, Chaitali Chakrabarti†, Achilleas Anastasopoulos*, Scott Mahlke* and Trevor Mudge*
*EECS Department, University of Michigan, Ann Arbor
†School of ECEE, Arizona State University, Tempe
{qizheng,yajchen,rdreslin,anastas,mahlke,tnm}@umich.edu, chaitali@asu.edu

***Abstract*—The rapid growth in the number of mobile devices and the higher data rate requirements of mobile subscribers have made wireless signal processing a key driving application of mobile computing technology. To design better mobile platforms and the supporting wireless infrastructure, it is very important for computer architects and system designers to understand and characterize the performance of existing and upcoming wireless protocols.**

In this paper, we present a newly developed open-source benchmark suite called WiBench. It consists of a wide range of signal processing kernels used in many mainstream standards such as 802.11, WCDMA and LTE. The kernels include FFT/IFFT, MIMO, channel estimation, channel coding, constellation mapping, etc. Each kernel is a self-contained configurable block which can be tuned to meet the different system requirements. Several standard channel models have also been included to study system performance, such as the bit error rate. The suite also contains an LTE uplink system as a representative example of a wireless system that can be built using these kernels. WiBench is provided in C++ to make it easier for computer architects to profile and analyze the system. We characterize the performance of WiBench to illustrate how it can be used to guide hardware system design. Architectural analyses on each individual kernel and on the entire LTE uplink are performed, indicating the hotspots, available parallelism, and runtime performance. Finally, a MATLAB version is also included for debugging purposes.

I. Introduction

The mobile market has experienced a rapid increase over the last decade. It is expected that by the end of 2013 there will be almost as many mobile-cellular subscriptions as there are people in the world [1]. The number of mobile broadband subscribers, who access the internet wirelessly through mobile devices, has climbed from 268 million in 2007 to 2.1 billion in 2013—a 40% annual increase rate [1]. To support this growth the number of base stations has also increased exponentially [2]. All indications show that this trend is likely to continue, at least in the near future.

In order to design better mobile platforms and the wireless infrastructure to support them, computer architects and system designers will have to understand and characterize the performance of wireless protocols. In a nutshell, wireless protocols encode the raw information in the transmitter side, and recover it in the receiver side. These processes consume significant computing resources and power in a handheld system. For instance, a GSM subsystem in a smartphone consumes 30%-50% of the overall power [3], and an even larger portion is used in more recent WCDMA and LTE protocols. In addition, the portion of the global CO_2 footprint for wireless networks will be 13% of the total allocation to information and communication technology (ICT) by 2020, according to the Climate Group [4]. Clearly it is important that wireless devices be power-efficient—requiring designers to understand the power/performance characteristics of the algorithms within these protocols.

Benchmarks are an important tool for characterizing power/performance tradeoffs in different application domains. Examples of important benchmark suites include SPEC benchmarks [5] for general purpose computing, PARSEC benchmarks [6] for multithreaded applications, MEVBench [7] for mobile computer vision applications, and BBench [8] for interactive smartphone applications. Although there exist some benchmarks for wireless communication, they either are out-of-date, lack essential algorithm details, or distorted the computational characteristics by introducing addition overhead.

In this paper, we develop an open source configurable kernel set for wireless signal processing called WiBench[1]. The set consists of important signal processing kernels that are widely used in many wireless standards such as 802.11, WCDMA or LTE. The kernels include Fast Fourier transform (FFT), multiple-input and multiple-out (MIMO) detection, channel estimation, channel coding, constellation mapping, and scrambling. Each kernel is a self-contained configurable block. Such a system can be used to build multiple wireless protocols and evaluate their performance. To demonstrate this feature, we include an LTE uplink benchmark in WiBench. LTE is a fourth generation wireless communication standard (4G) that is being deployed worldwide. It is designed to deliver data rates up to 100 Mbps. The configurability of WiBench kernels allows our LTE uplink to support a variety of peak data rates ranging from 1.56 to 100 Mbps. We also include several standard channel models in WiBench so that system researchers can use it to evaluate the bit error rate (BER) performance of their system. WiBench is provided in C++, which enables architecture researchers to characterize applications, and MATLAB, which helps debugging and functional verification.

Our key contributions are:

- An open source configurable wireless signal processing kernel suite, which includes a rich set of key signal processing kernels that are used widely in mainstream wireless protocols.

[1]WiBench is available through http://wibench.eecs.umich.edu.

- An LTE uplink in the benchmark that illustrates how to build a wireless application by assembling kernels. The configurability of our kernels allows us to support different peak data rates. Users can similarly establish their own applications to model WCDMA or Wi-Fi.
- Benchmark support for several standard channel models that allows system designers to evaluate their decisions by examining BER.
- A demonstration of WiBench for hardware design which analyses and identifies the hotspots, available parallelism, and runtime performance at the kernel and system levels.

The rest of paper is organized as follows. Section II presents the related work. In Section III, we describe our kernel suite, and provide details of each kernel in the benchmark and the LTE uplink. We also explain our design philosophy of WiBench. In Section IV, we examine the characteristics of each individual kernel and the LTE uplink, and provide pointers for efficient hardware design.

II. Related Work

The wireless communication community works on open problems in telecommunication as well as next generation technologies. Their primary focus is on the impact of communication theory and algorithm optimization on system performance, typically measured in terms of BER. There are many open source system simulators written in MATLAB or built through Simulink [9]–[11] to aid in this analysis. MATLAB and Simulink are easy to use due to their interactive natures and a large number of built-in functions. However, MATLAB and Simulink are not suitable for hardware design because their abstraction levels are too high. To aid in the co-design of systems and their underlying architecture, we release both MATLAB and C++ versions of all kernels in WiBench—System designers may explore BER through MATLAB, and architects can explore power-efficient hardware organizations that execute the C++ code. While most system simulators are in MATLAB/Simulink, there are several that include C/C++ versions which will be discussed in the following paragraphs.

First, the closest related work to WiBench is GNU Radio [12], a free and open source software toolkit providing signal processing blocks for software radio implementation. GNU Radio uses a "block" abstraction to connect signal processing kernels, which are implemented in C++, together with a few lines of Python code. Each block is equipped with its own input/output buffers. The GNU Radio suite then uses a runtime scheduler that activates each block when there is enough data in its input buffer and space in its output buffer to perform the function. It is designed to run on commodity hardware. To construct a complete end-to-end wireless system, the user must first understand algorithmic details of these blocks. WiBench has a different goal, which is to support hardware exploration of domain specific hardware solutions. To this end, we provide all key kernels as well as the entire system in WiBench. In addition, the GNU radio block class introduces non-kernel overheads. This may lead to a distorted picture of how the signal processing algorithms would perform on domain specific hardware. WiBench's behavior is closer to the actual computational characteristics of wireless signal processing kernels, similar to the approach used in the design of several high-performance DSP prototypes [13]–[15].

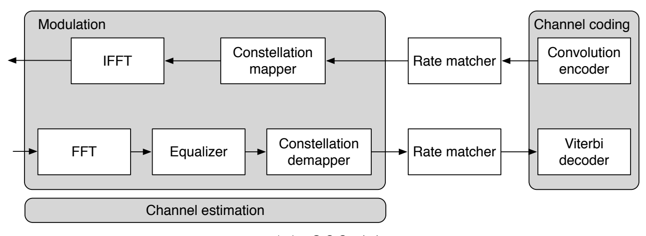
Second, MiBench [16] is a set of embedded applications released over a decade ago. Telecommunication, one of the six categories in the benchmark, contains GSM related processing—FFT/IFFT, GSM voice encoding and decoding algorithms, Adaptive Differential Pulse Code Modulation encode/decode, and CRC32 checksum algorithm. These represent only a small portion of wireless signal processing kernels. Since the release of MiBench in 2001, communication technology has seen rapid development including several generations of technology enhancements rendering many of the MiBench kernels irrelevant. WiBench is designed specifically for wireless signal processing and includes many state-of-the-art algorithms that will be used in next generation technology.

Third, LTE Uplink Receiver PHY Benchmark [2] is an open source, freely available benchmark that represents the baseband processing of an LTE base station. The benchmark implements SC-FDMA modulation, channel estimation, transform decoding and soft symbol demapping, and is capable of generating different number of users with different workloads. However, this benchmark mainly aims to simulate the workload change in an LTE base station to study the power management strategy, rather than the characterization of wireless algorithms for hardware design. In addition, it only includes some parts of the LTE uplink and is missing the details of several important kernels, for example the Turbo decoder is represented simply as a sleep function. Ultimately this limits the use of this benchmark in a wider scope of wireless system design. Our WiBench contains all the signal processing kernels for LTE in both MATLAB and C++ versions.

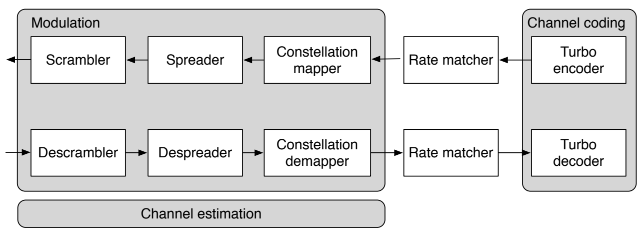
Finally, the BDTI^{TM} OFDM receiver benchmark [17] is a commercial benchmark for evaluating multi-core and other high-performance processing engines for communication applications. Public information about this benchmark is limited, but their website does indicate that they still use the Viterbi decoder rather than the more state-of-the-art Turbo decoder present in WiBench. The BDTI^{TM} OFDM receiver benchmark requires a license for use, in contrast to WiBench.

TABLE I: The components of WiBench

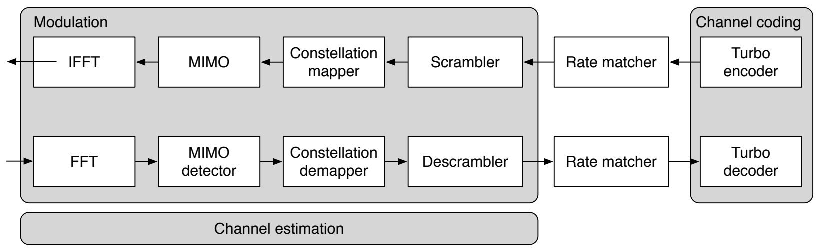
Category	Benchmark
Kernels	Channel coding/decoding
	Rate matching
	Scrambling/Descrambling
	Constellation mapping/demapping
	MIMO detection
	FFT/IFFT
	Sub-carrier mapping/demapping
	Channel Estimation
Channel models	Gaussian Random Channel model (GRC)
	Extended Pedestrian A model (EPA)
	Extended Vehicular A model (EVA)
	Extended Typical Urban model (ETU)
Applications	LTE uplink



(a) 802.11a



(b) WCDMA



(c) LTE

Fig. 1: **The downlink flow charts of 802.11a, WCDMA and LTE [13].** This figure shows that different wireless systems have many common signal processing kernels, such as FFT/IFFT, channel coding, constellation mapping, etc. We picked the most frequently used algorithms to include in our benchmark.

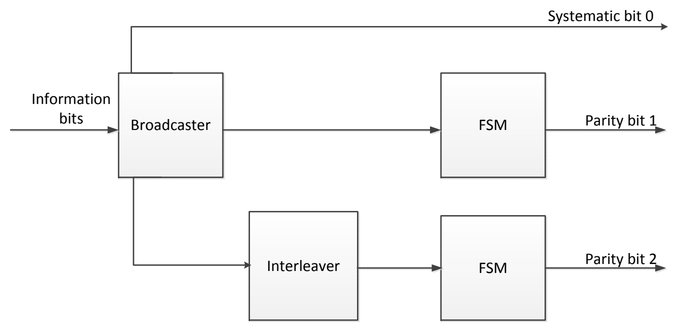


Fig. 2: **The structure of the Turbo code encoder.** The Turbo encoder consists of two FSMs and an interleaver. The outputs of the encoder are the original input sequences interleaved with outputs of two FSMs.

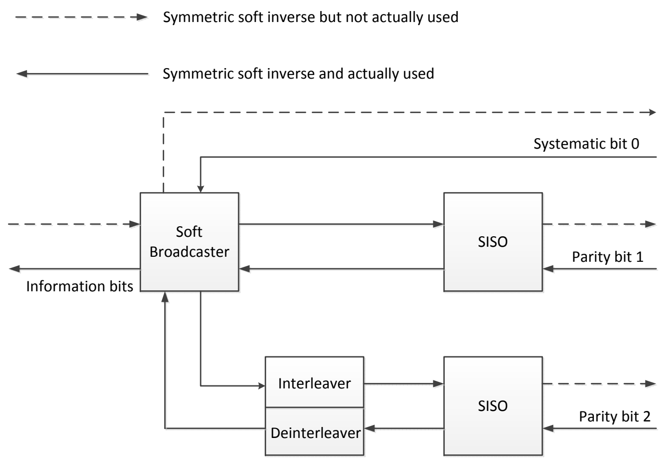


Fig. 3: **The structure of the Turbo code decoder.** It consists of two Soft-Input-Soft-Output decoders and an interleaver. The decoder works in an iterative fashion.

III. Benchmark Description

A. Design Philosophy

WiBench was built to handle multiple wireless protocols. Thus, unlike some recent benchmarks [2], WiBench was constructed with configurable kernels, which are the basic blocks for multiple wireless systems. The intent is for users of current and possibly future wireless systems to be able to design their own system using these building blocks and characterize them. Figure 1 illustrates the downlink flow charts of several mainstream wireless systems. It shows that different wireless systems actually share a lot of common signal processing kernels. In this work, we selected kernels that are most frequently used and are representative of the algorithms that are used in many wireless protocols. We also include several standard channel models so that system designers can test the performance of their systems under different channel conditions. Additionally, we show users how to use these kernels to build their own wireless systems by including an LTE uplink system in the benchmark. Table I summarizes the details of the benchmark. WiBench is originally written in C++, but we also provide a MATLAB version to facilitate debugging and functional verification.

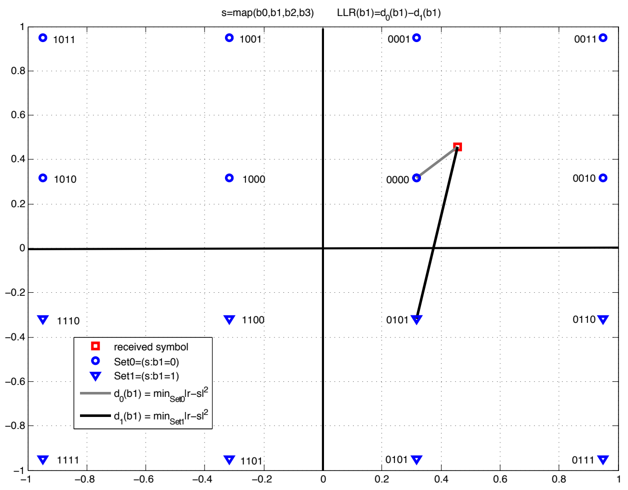


Fig. 4: **The constellation demapping of 16QAM.** In constellation mapping, every four binary bits are mapped to one of the sixteen complex values (circles and triangles). In constellation demapping, the distances between a received symbol (square) and all sixteen complex values (circles and triangles) is computed and the distances are used to recover the four bits of data.

B. Kernels

1) Channel coding: Channel coding is the technique used to control errors in data transmission over noisy channels that enable reliable delivery of digital data. There are many different channel coding techniques such as convolutional codes [18], Turbo codes [19], and Low Density Parity Check codes (LDPC) [20], etc. The Turbo codes we chose belong to a high-performance forward error correction family of codes widely used in 3G/4G mobile communications.

The scheme of our Turbo encoder is a Parallel Concatenated Convolutional Code (PCCC) with two Finite State Machines (FSM) and one internal interleaver. The structure of the Turbo encoder for $R = 1/3$ code is shown in Figure 2; One information bit is encoded into three transmitted bits.

The Turbo decoder architecture includes two Soft-Input-Soft-Output (SISO) decoders [21] and one internal interleaver/deinterleaver as illustrated in Figure 3. Inside each SISO decoder, a forward and backward trellis traversal algorithm is performed [21]. The Turbo decoder works in an iterative fashion—increasing the iteration number results in a better error correction performance at the cost of higher computation. Our Turbo code implementation supports 188 different input lengths from 40 to 6144.

2) Rate matching: The purpose of rate matching is to provide a variety of channel coding rates from a single "mother code" with a fixed rate R. This considerably increases the flexibility of a system in terms of the performance-complexity tradeoff of channel coding. Rate matching is performed by puncturing or by repeating coded bits. Internally, the rate matching algorithm buffers the incoming bit stream and does bit collection, selection and pruning.

3) Scrambling/Descrambling: Scrambling encrypts and randomizes data. It encodes the transmitted information to make it unintelligible to a potential eavesdropper. The bit stream in a subframe is scrambled with a User Equipment (UE) specified scrambling sequence in the transmitter, which is reversed by descrambling at the receiver side. Our implementation supports arbitrary lengths of scrambling.

4) Constellation mapping/demapping: The goal of constellation mapping is to represent a binary data stream with a signal that matches the characteristics of the channel [22]. The binary sequences are grouped and mapped into complex-valued constellation symbols. Figure 4 shows a 16 Quadrature Amplitude Modulation (16QAM) constellation, where every four bits are mapped to one of the sixteen complex values (circles and triangles in Figure 4). We implemented BPSK (1-bit Constellation), QPSK (2-bit Constellation), 16QAM (4-bit constellation), and 64QAM (6-bit constellation) mapping in our benchmark.

Constellation demapping retrieves the binary stream from the signal by generating either hard or soft information. Hard information selects and outputs the binary representation of the closest symbol to the received signal (e.g., (0000) in the example of Figure 4). Soft information computes likelihood ratios for each bit that will be used by the channel code decoder as bit metrics. Figure 4 interprets the process of generating logarithmic likelihood ratios (LLRs) for the second bit (i.e., bit $b1$) of a received symbol r.

5) Multiple-Input Multiple-Output (MIMO): MIMO technology is the use of multiple antennae at both the transmitter and the receiver with the aim of increasing performance and/or data rate. *MIMO for spatial multiplexing* transmits independent data streams from each of the multiple transmit antennae, thus increasing the system data rate. *MIMO for diversity* transmits a single data stream from each of the multiple transmit antennae. The single data stream is coded by space-time coding, which improves the reliability of data transmission. There are various MIMO detection methods, for example, linear detection, sphere decoder, lattice reduction detection, etc. We include some widely used algorithms in WiBench, including a Least Square (LS) based zero forcing detection and a tree based sphere decoder. Our MIMO detection module includes different antenna configurations including 1×1, 2×2 and 4×4.

6) FFT/IFFT: Discrete Fourier Transform (DFT) is one of the most frequently used transformations in science and engineering. It transforms a finite set of samples of a function in the time domain into frequency domain; inverse IDFT reverses this operation. Fast Fourier transform (FFT) is a fast algorithm to compute DFT. It requires only $O(NlogN)$ operations to get the same result as DFT. We utilized FFTW [23] to implement FFT/IFFT. FFTW is a C library for computing the DFT that adapts to the running hardware platform to maximize performance. Its performance is competitive with, or even better than, some highly-tuned FFT implementations such as Suns Performance Library and IBMs ESSL library [24]. Our kernel supports any FFT/IFFT size in the form of $2^a \cdot 3^b \cdot 5^c \cdot 7^d$.

7) Sub-carrier mapping/demapping: The mapping kernel inserts data and reference symbols into the sub-carrier. If multiple users exist in the system, their data will be mapped into non-overlapping sub-carriers. The demapping kernel extracts data and reference symbols from the sub-carrier for each user in the system.

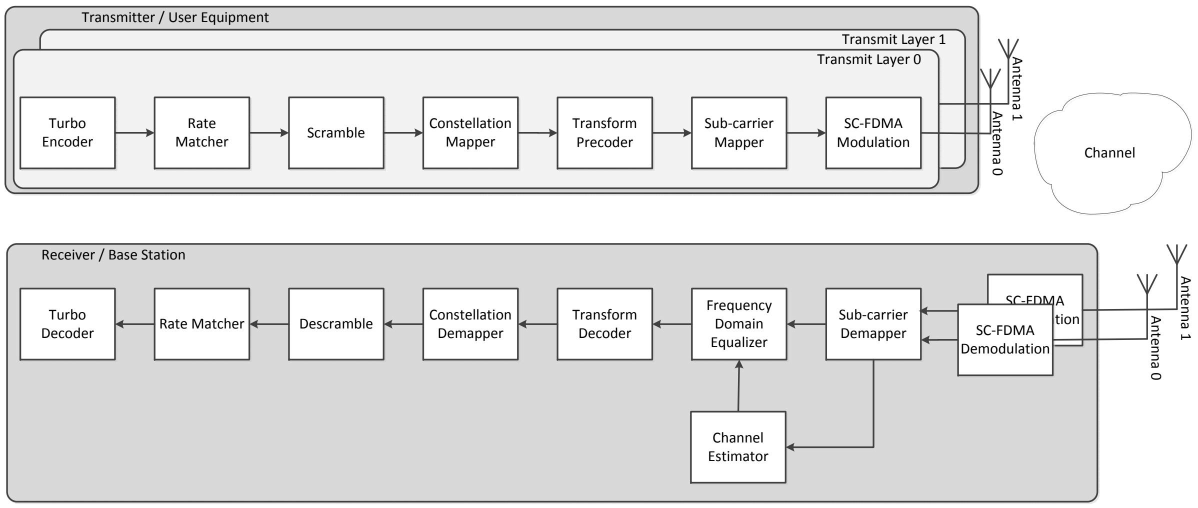


Fig. 5: **The system flow graph of the LTE uplink.** It contains the Turbo coding, rate matching, scrambling, constellation mapping, transform precoding, sub-carrier mapping, SC-FDMA modulation, channel estimation and equalization.

8) Channel Estimation: In order to achieve reliable communication most kernels in the receiver side require knowledge of the channel parameters, also known as Channel State Information (CSI) [25]. CSI can be obtained in two ways. One is to insert known symbols as pilots into data sequences, and the performance of pilot signals is used for estimation. The other is blind estimation by using knowledge of statistical characteristics of the received signal. Most blind methods suffer from several drawbacks such as slow convergence speed, high complexity, and poor performance. As a result pilot aided channel estimation is more common, therefore, we adopt it for WiBench. There is a choice of algorithms for pilot aided channel estimation, including Least Square (LS) estimation and Minimum Mean Square Error (MMSE) estimation. MMSE estimation provides better performance than LS, but requires more computation and sophisticated statistical characteristics of the channel. We include both LS and MMSE kernels in the benchmark.

C. Channel Models

The channel model represents the characteristic degradation of the signal as it is transmitted wirelessly through the environment. In order for system designers to measure the BER that a particular receiver configuration experiences there are several standardized channel models. The basic channel model is a Gaussian random channel (GRC) which introduces Gaussian noise to the signal. In addition to the GRC, we include several other channel models—Extended Pedestrian A model (EPA), Extended Vehicular A model (EVA), and Extended Typical Urban model (ETU) [26]—which provide more realistic channel scenarios.

D. Application: LTE uplink

We built an LTE uplink system to illustrate how to use the kernel and channel models provided in WiBench to create a complete wireless link. Similarly, other systems such as WCDMA and 802.11a can also be built. The LTE uplink system is organized as shown in Figure 5. We implemented the entire physical layer as well as the most compute-intensive parts of the transport layer including the Turbo decoder and rate matching. Our LTE uplink supports configurations covering all transmission bandwidths whose peak data rate ranges from 1.56 to 100 Mbps. In Section IV we will evaluate the performance of each kernel in the LTE uplink and show an example system analysis by determining the BER under different channel conditions. In the following subsections we describe in more detail the specific kernel choices for the LTE application.

1) Turbo encoder/decoder: The FSM of the Turbo encoder in the LTE specification is an 8-state recursive systematic convolutional encoder [27]. For our analysis, we set the iteration number of the Turbo decoder at 5. Although we have fixed the iteration number, WiBench could be used to explore the tradeoff between BER performance and the amount of computation for different numbers of iterations.

2) Single Carrier Frequency Diversity Multiple Access (SC-FDMA): SC-FDMA is a precoded Orthogonal Frequency Diversity Multiplexing (OFDM) scheme, which has an additional transform precoding step that precedes the conventional OFDM processing. OFDM processing encodes data on multiple carrier frequencies. OFDM is applied in the LTE downlink (base station to user equipment), while SC-FDMA is realized in the uplink (user equipment to base station). Compared to OFDM, SC-FDMA has two main advantages that are critical to the uplink transmission: 1) SC-FDMA has a lower Peak-to-Average Power Ratio; 2) SC-FDMA is less sensitive to frequency offsets than OFDM.

In the transmitter, we implement the OFDM step of SC-FDMA by performing IFFT and inserting a Cyclic Prefix (CP). In the receiver, we eliminate the inter-symbol interference by removing CPs and converting data from the time domain to the frequency domain by FFT. The transform precoding step of SC-FDMA is done with a $2^a \cdot 3^b \cdot 5^c$ mixed radix FFT, while the IFFT is performed in the transform decoder at the receiver side.

3) Channel estimation: The LTE uplink transmission uses the comb-type pilot arrangement [28], where only time domain interpolation needs to be applied. The uplink pilot reference symbols from different transmit antennae occupy the same sub-carriers. However, pilot reference symbols are designed so that they can be distinguished from each other at the receiver side. Channel estimation takes the received signal and known pilot reference symbols to estimate the CSI, which is then used to compute the channel coefficients. We selected the frequency domain least square estimator that provides an acceptable performance with reasonable computation under the assumption that we have no knowledge of the channel [29].

4) Equalizer: The equalizer we apply is a zero forcing MIMO detector in the frequency domain. Taking advantage of OFDM/SC-FDMA, channel equalization in LTE can be implemented simply by a Frequency Domain Equalizer (FDE) with the coefficients estimated by the channel estimator.

IV. Benchmark Characterization

We perform four studies to characterize the benchmark suite on two different types of processors, illustrating how WiBench can be used for hardware design and system study. First, we profiled each individual kernel, determining how each performs on different processors. This type of analysis can be used by hardware architects to design the underlying hardware to achieve power-efficient systems and by code designers to better target optimization points. Second, we explore the performance of the LTE uplink included in the benchmark for different bandwidth requirements. Third, we show how different LTE uplink configurations with the same bandwidth impact the relative importance of each kernel. Finally, we perform an analysis of how the LTE uplink performs, in terms of BER, under one type of channel conditions. This type of analysis can be used by system designers to explore how their design performs under different channel conditions.

A. Experimental Setup

We performed our analyses on cores that are used in desktop systems and embedded devices, because wireless applications run on both embedded platforms (e.g. smartphones) and server-like machines (e.g. wireless base stations). For the desktop class processor, we used an Intel Core i7-2600 CPU running Linux 3.2.0-38-generic. For the embedded system, we utilized an NVIDIA ION box with an Intel Atom 330 processor and 4 GB of SDRAM. The Intel Atom is the Intel's line of low-power, low-cost microprocessors [30], whose SoC platform is used in many smartphones and tablets such as Lenovo K800, Motorola RAZR i, Safaricom Yolo, Samsung Series 5 Slate, and HP ElitePad 900 [31], [32]. The detailed configuration of the systems are presented in Table II. The benchmarks were compiled using GNU g++ compiler suite version 4.6.3 with O2-level optimization. Intel VTune Amplier XE 2013 was used to gather code hotspot information and instructions per cycle (IPC) for the wireless benchmarks. VTune Amplifier XE is a performance profiler provided by Intel for x86 based processors. It provides information on code performance, including the hotspots, CPU utilization, multithread synchronization overhead, etc.

B. Individual Kernel Characterization

For the first example study we analyse the performance of each kernel in WiBench. Table III describes the configurations

TABLE II: System configurations of the profiling platforms

Feature	Configuration	
	Desktop platform	Mobile platform
Operating System	Linux 3.2.0-38	Linux 3.2.0-39
Processor	Intel Core i7 2600	Intel Atom 330
Frequency	3.40 GHz	1.60 GHz
L1 I-Cache	32 KB	32 KB
L1 D-Cache	32 KB	24 KB
L2 Cache	256 KB	512 KB
Last Level Cache	8 MB	N/A
Memory	16 GB DDR3	4 GB SDRAM
Out-of-order	Yes	No
Single core issue width	4	2
SIMD	128-bit, SSE2, SSE3, SSSE3, SSE4	128-bit, SSE2, SSE3, SSSE3

TABLE III: The configurations of the individual kernel

Kernel	Configuration
Turbo decoder	code rate $= 1/3$, codeword length $= 1184$
Descrambling	sequence length $= 300$
Constellation demapping	QPSK, sequence length $= 150$
FFT	128
IFFT	75
MIMO	2×2, sequence length $= 75$

of each kernel. Figure 6 compares the IPCs obtained by the two platforms. Based on Figure 6, the i7 processor, with dynamic out-of-order scheduling, can make use of instruction level (ILP) and memory level parallelism (MLP) in order to issue more instructions per cycle than the Atom processor, even taking the issue width difference into account . Since out-of-order execution requires more complex hardware, leading to a high power consumption, this improvement must be balanced against the limited power budget of embedded platforms.

Next we study the speedup obtained by compiling each kernel with automatic vectorization flags. We used -ftree-vectorize -msse2 -ffast-math for automatic vectorization; the corresponding results are shown in Figure 7. When the automatic vectorization is enabled, SIMD instructions are inserted automatically by the compiler. By using automatic vectorization, we can get as much as 1.45× speedup on the i7 and 1.85× speedup on the Atom. However, since automatic vectorization is implemented by the compiler, there is a limited range over which it works. Table IV shows the theoretical SIMD width of each kernel obtained by analyzing the code manually. The kernels do not achieve this speedup when using automatic vectorization either because there is no vectorizable operation in the kernel, or it is difficult for the compiler to extract the parallelism. More speedup is expected if the program is vectorized manually using SIMD intrinsics. Overall, the results indicate that hardware platforms designed for these kernels should include vectorization support and that hand optimized

TABLE IV: The theoretical SIMD width of individual kernels for the configurations in Table III

Kernel	SIMD width
Turbo decoder	8
Rate matching	1
Descrambling	300
Constellation demapping	600
LS detection	150
Tree-based detection	300
FFT	128
IFFT	75
Channel estimation	300

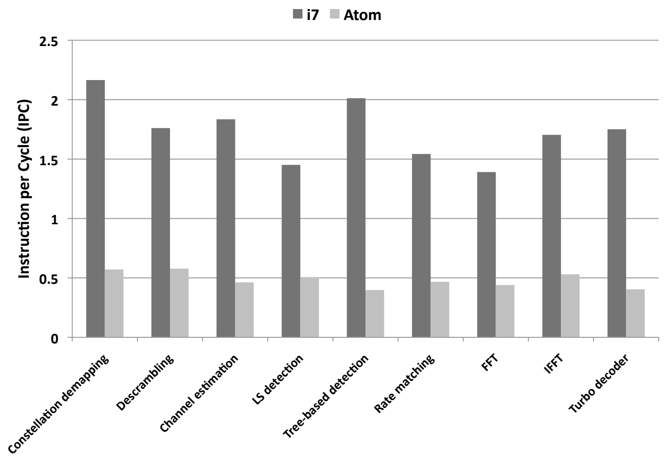


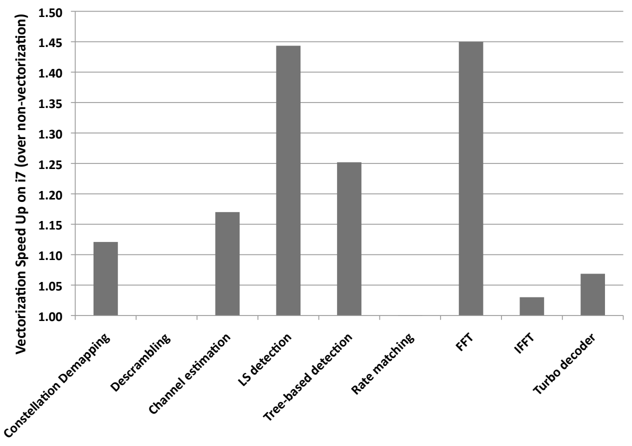
Fig. 6: **IPCs for desktop and embedded processors.** The IPCs of kernels on the i7 processor are higher than those on the Atom processor even taking the issue width difference into account. Because the i7 is an out-of-order processor, it can dynamically schedule instructions and take advantage instruction and memory level parallelism.

code/libraries will continue to be needed in order to attain better performance.

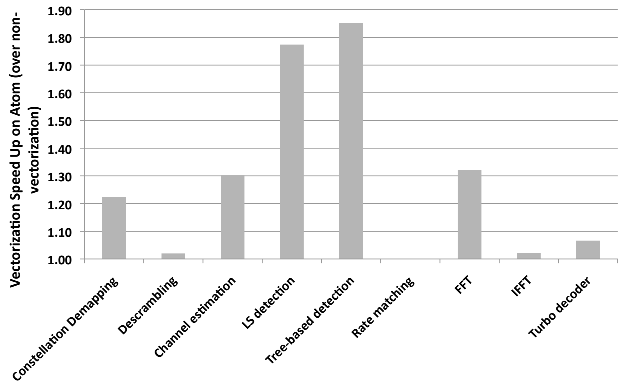
C. Application Example: LTE Uplink System

Next we profile the LTE Uplink provided in the benchmark with respect to hotspots and runtime performance. Because most of the computations are done in the receiver side, we only profile kernels in the LTE uplink receiver. We perform four studies: 1) a characterization of one LTE uplink configuration across different peak data rates; 2) a characterization of different LTE uplink configurations for a fixed peak data rate; 3) an analysis on the sizes of data transfered between kernels; and, 4) a study of the BER for the LTE uplink under a Gaussian Random Channel model.

1) LTE Uplink Characterization: We first studied the breakdown of runtime for the LTE uplink to determine the computational hotspots. For the LTE uplink, we used peak data rates varying from 1.56 to 100 Mbps to assess the runtime changes of the algorithms. Figure 8 shows the time spent by each kernel as a fraction of the overall system runtime for both the i7 and the Atom platforms at 100Mbps (the system



(a) i7 processor



(b) Atom processor

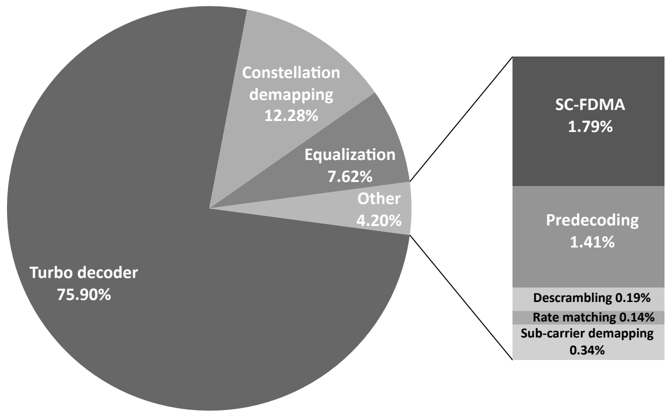
Fig. 7: **Vectorization Impact on (a) i7 and (b) Atom for configurations in Table III.** These graphs show speedups achieved when kernels were compiled with automatic vectorization flags turned on. The results suggest that hardware platforms should include vectorization support when running these kernels.

TABLE V: The configurations of the LTE uplink at 100Mbps

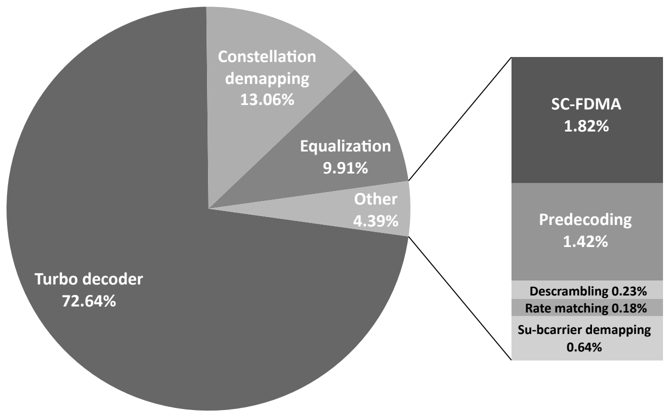
Kernel	Configuration
Turbo decoder	code rate $= 1/3$, codeword length $= 6144$
Constellation demapping	16QAM
FFT	2048
IFFT	1200
MIMO	2×2

configuration is shown in Table V). We see that the Turbo decoder takes more than 70% of the execution time. Thus, for high throughput applications, either the Turbo decoder should be highly optimized for the specific platform or it should be implemented by a hardware accelerator.

Next we measured the total runtime of the LTE uplink for different subframe sizes to illustrate the performance of the system as the workload changes. Figure 9 demonstrates that the processing time of an LTE uplink subframe increases pro-



(a) i7 processor



(b) Atom processor

Fig. 8: **Breakdowns of the LTE uplink runtime among the kernels on (a) i7 and (b) Atom.** The results indicate that hardware designers should put much concern on expediting the Turbo decoder. It should be either highly optimized for the specific platform or implemented by a hardware accelerator.

portionally to the subframe size. Since the size of a subframe is proportional to the system peak data rate, the processing time of an LTE uplink subframe is also proportional to the system peak data rate. This indicates that the dynamic operation count for most LTE uplink kernels scale linearly.

TABLE VI: The configurations of the LTE uplink at 12.5 Mbps.

Config	FFT	IFFT	MIMO	Constellation Demapping
A	256	150	2×2	16QAM
B	512	300	1×1	16QAM
C	512	300	2×2	QPSK
D	1024	600	1×1	QPSK

2) Different LTE Configurations: Next we looked into the runtime changes of each individual kernel for different LTE uplink configurations. For this study, the LTE uplink peak data rate is fixed at 12.5 Mbps. While the 12.5Mbps can be achieved by many configurations, in this study we choose four representative configurations, presented in Table VI. These configurations differ in the size of the OFDM symbol, number of antennas and constellation size. For instance, a large FFT configuration with simpler constellation can be used for bad channel conditions with larger bandwidth usage, while a small FFT configuration with complex constellation and more antennae can be used for good channel condition but limited bandwidth. We assume that the Turbo decoder is implemented by a specialized accelerator and exclude the Turbo decoder runtimes. This is a reasonable assumption because these accelerators are typical even in programable wireless signal processors. Figure 10 shows the results. From the figures, we derive the following conclusions.

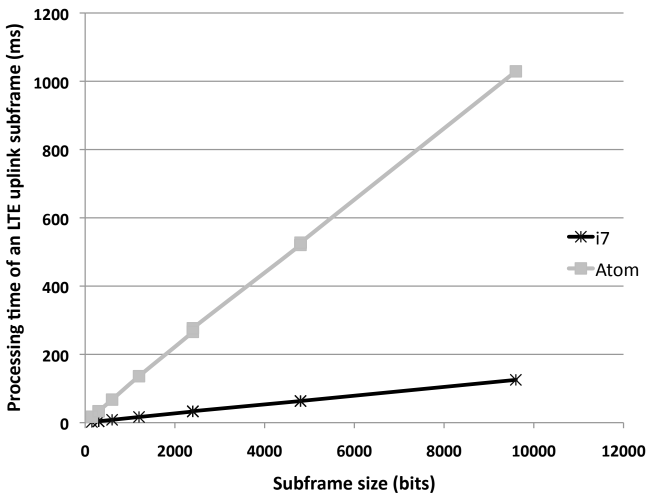
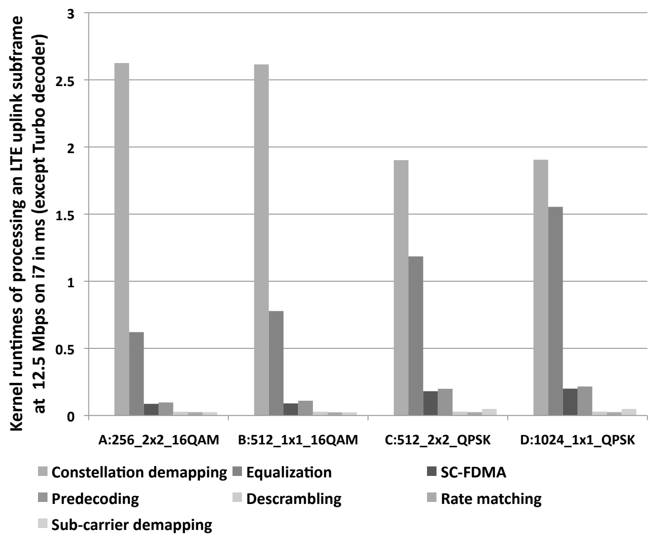


Fig. 9: **Processing times of an LTE uplink subframe with different subframe sizes.** The larger the subframe size, the higher the peak data rate. It shows that the processing time of an LTE uplink subframe is proportional to the subframe size. This indicates a linear scaling of the dynamic operation count for most LTE uplink kernels.

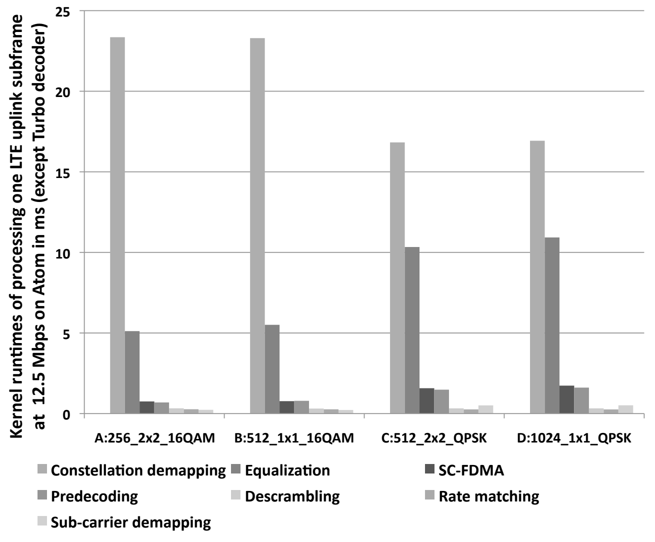
- The constellation demapping and equalization kernels take most of the execution time (when excluding Turbo) for all four configurations. Therefore, hardware and software optimizations should be done to accelerate these two kernels.
- The importance of each kernel changes as the system configuration changes, even if the data rate remains the same. While constellation demapping is much more important than all the other kernels, equalization, FFT and IFFT are also important for Configuration D.

3) Data transfer between kernels: In this study, we look at how much data is transfered between different kernels. Figure 11 demonstrates the data movement between kernels when processing one LTE subframe for the configuration in Table V. The values in the red circles represent the amount of data movement, which indicates the minimum sizes needed for the buffers containing the intermediate results between adjacent kernels. Because on-chip memory is an expensive resource, this information helps domain specific computer architects design their memory system.

4) Exploring Channel Models and BER: This study shows an example of how system designers can connect an entire LTE uplink out of the kernels and connect them through a



(a) i7 processor



(b) Atom processor

Fig. 10: **The runtimes of kernels (exclude the Turbo decoder) in LTE uplink with different configurations at 12.5 Mbps on (a) i7 and (b) Atom processor.** The results suggest hardware and software optimizations on the constellation demapping and equalization kernels. The graph also demonstrates the importance change of each kernel as the system configuration changes.

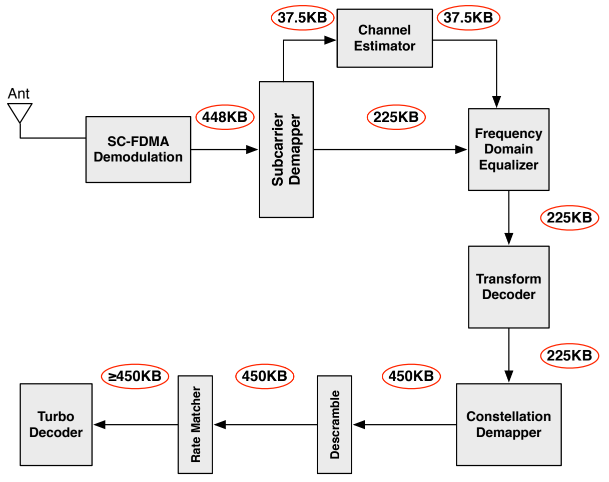


Fig. 11: **The sizes of data movement between kernels.** The results show how much data needs to be stored in the buffers for the intermediate results between adjacent kernels to process one LTE subframe for the configuration in Table V.

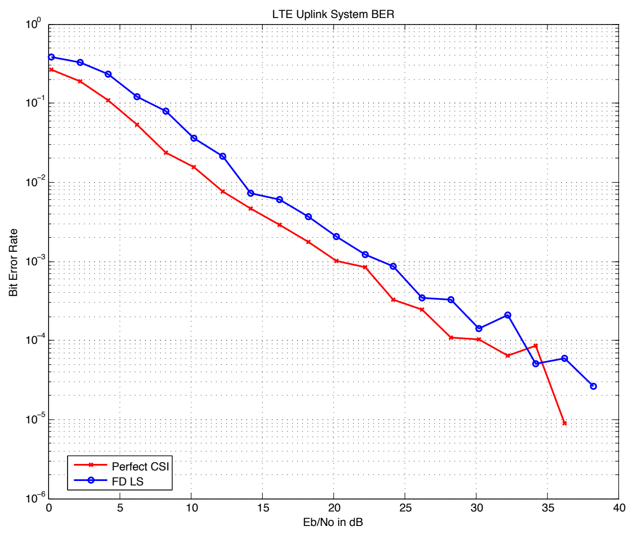


Fig. 12: **BER of LTE uplink through Gaussian random channel with AWGN.** The configurations are: FFT This graph shows the BER performance of the LTE uplink system under a Gaussian random channel with AWGN.

channel model and inject noise to measure BER. The kernel configurations are the same as those in Table III. We studied the BER performance of our LTE uplink system under a Gaussian random channel (the amplitude follows a Rayleigh distribution) with additive white Gaussian noise (AWGN). The BER performance is shown in Figure 12. BER is calculated by collecting the difference between the information bits encoded in the transmitter and those decoded at the receiver end. Perfect CSI means that the receiver knows the exact channel impulse response when processing received data. The FD LS curve expresses the performance of a system running with a frequency domain least square channel estimator, which is a more realistic scenario.

D. Architectural Implications from LTE characterization

The above analyses show that the Turbo decoder, constellation demapping, and equalization are the most important kernels in an LTE uplink. Thus a processor designed for LTE uplink should efficiently execute these three kernels. Second, constellation demapping and equalization (consisting of MIMO detection and channel estimation) have very large theoretical SIMD widths (from Table IV), and also achieve appreciable speedups with automatic vectorization by the compiler. Therefore, a wide SIMD engine should be included in the processor to accelerate these two kernels. In contrast, the Turbo decoder has a small SIMD width and little speedup with automatic vectorization. Furthermore, the Turbo decoder takes the largest portion of the runtime, suggesting it should be mapped to a hardware accelerator—which is often the case in today's practice. Finally, we noted that the importance of each kernel varies when system configurations are different. All together, these observations illustrate the usefulness of a benchmarking infrastructure to evaluate wireless signal processing systems.

V. Conclusion

As the mobile market continues to grow rapidly wireless signal processing is becoming one of the primary uses of computing technology. Consequently, closer attention is being paid to the hardware platform design and its power consumption. Computer architects usually benefit a great deal from analyzing application benchmarks during design time to gain insight into power and performance tradeoffs. In this paper we presented an open source benchmark suite of wireless system kernels and channel models to support hardware and system design of wireless signal processing platforms. We characterized the benchmark suite on two different types of processors to illustrate how it can be used. Users can easily build their own wireless systems by simply assembling our kernels together to realize a target configuration.

VI. Acknowledgement

We thank the anonymous reviewers for their efforts and helpful suggestions. We also wish to thank Nilmini Abeyratne, Joseph Pusdesris and Jui Wu for their generous and useful help on WiBench and the paper. This work is in part supported by NSF-CNS-0910851 and ARM.

References

[1] "The World in 2013: ICT Facts and Figures," International Telecommunication Union, 2012.

[2] M. Sjalander, S. McKee, P. Brauer, D. Engdal, and A. Vajda, "An LTE Uplink Receiver PHY Benchmark and Subframe-based Power Management," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2012, pp. 25–34.

[3] A. Carroll and G. Heiser, "An Analysis of Power Consumption in a Smartphone," in *USENUX*, 2010.

[4] "SMART 2020: Enabling the Low Carbon Economy in the Information Age," The Climate Group on behalf of the Global eSustainability Initiative (GeSI), 2008. [Online]. Available: http://www.smart2020.org/_assets/les/02_Smart2020Report.pdf

[5] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta, "Complete Computer System Simulation: the SimOS Approach," *IEEE Parallel Distributed Technology: Systems Applications*, vol. 3, no. 4, pp. 34–43, 1995.

[6] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proceedings of the 17th International Conference on Parallel architectures and Compilation Techniques (PACT)*, 2008, pp. 72–81.

[7] J. Clemons, H. Zhu, S. Savarese, and T. Austin, "MEVBench: A Mobile Computer Vision Benchmarking Suite," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2011, pp. 91–102.

[8] A. Gutierrez, R. Dreslinski, T. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system Analysis and Characterization of Interactive Smartphone Applications," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2011, pp. 81–90.

[9] MATLAB Central–File Exchange. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/

[10] X. Guo and P. Song, "Simulink Based LTE System Simulator," M. Sci. thesis, Chalmers University of Technology, Goteborg, Sweden, 2010.

[11] "LTE PHY Downlink with Spatial Multiplexing." [Online]. Available: http://www.mathworks.com/help/comm/examples/lte-phy-downlink-with-spatial-multiplexing.html

[12] "GNU Radio." [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki

[13] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "SODA: A High-Performance DSP Architecture for Software-Defined Radio," *IEEE Micro*, vol. 27, no. 1, pp. 114–123, 2007.

[14] M. Woh, Y. Lin, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, R. Bruce, D. Kershaw, A. Reid, M. Wilder, and K. Flautner, "From SODA to Scotch: The Evolution of a Wireless Baseband Processor," in *41st IEEE/ACM International Symposium on Microarchitecture (MICRO-41)*, 2008, pp. 152–163.

[15] M. Woh, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "AnySP: Anytime Anywhere Anyway Signal Processing," *IEEE Micro*, vol. 30, no. 1, pp. 81–91, 2010.

[16] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," in *IEEE International Workshop on Workload Characterization (WWC-4)*, 2001, pp. 3–14.

[17] "BDTI^{TM} OFDM Receiver Benchmark." [Online]. Available: http://www.bdti.com/Services/Benchmarks/OFDM

[18] A. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[19] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Intell. Transport. Syst.*, vol. 20, pp. 284–287, Mar. 1974.

[20] R. Gallager, "Low-density Parity-check Codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[21] Y. Lin, S. Mahlke, T. Mudge, C. Chakrabarti, A. Reid, and K. Flautner, "Design and Implementation of Turbo Decoders for Software Defined Radio," in *IEEE Workshop on Signal Processing Systems Design and Implementation (SIPS)*, 2006, pp. 22–27.

[22] J. G. Proakis and M. Salehi, *Digital Communications*, 5th ed. New York, NY: McGraw-Hill, 2008.

[23] M. Frigo and S. G. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[24] M. Frigo and S. Johnson, "FFTW: An Adaptive Software Architecture for the FFT," in *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 3, 1998, pp. 1381–1384 vol.3.

[25] "Wiki papge: Channel state information." [Online]. Available: http://en.wikipedia.org/wiki/Channel_state_information

[26] *LTE Specification*, 3GPP Std. 36.521.

[27] *LTE Specification*, 3GPP Std. 36.312.

[28] *LTE Specification*, 3GPP Std. 36.211.

[29] S. Coleri, M. Ergen, A. Puri, and A. Bahai, "Channel Estimation Techniques based on Pilot Arrangement in OFDM Systems," *IEEE Transactions on Broadcasting*, vol. 48, no. 3, pp. 223–229, 2002.

[30] List of Intel Atom microprocessors. [Online]. Available: http://en.wikipedia.org/wiki/List_of_Intel_Atom_microprocessors

[31] Atom (system on chip). [Online]. Available: http://en.wikipedia.org/wiki/Atom_(system_on_chip)

[32] Smartphones with Intel Inside. [Online]. Available: http://www.intel.com/content/www/us/en/smartphones/smartphones.html