Quantitative Performance and Power Analysis of LTE using High Level Synthesis

Yun Liang, Shuo Wang
Center for Energy-Efficient Computing and Applications, School of EECS, Peking University, China
Email: {ericlyun, shvowang}@pku.edu.cn

Abstract: Nowadays, IoT devices are ubiquitous in our daily life. 3GPP Long Term Evolution (LTE) protocol is the backbone supporting the IoT ecosystem. Moreover, IoT-friendly LTE requires LTE chip to be both high performance and energy efficient. In this paper, we propose to use high level synthesis to design LTE chips. We develop a framework to explore the design space of LTE kernels with respect to different performance and power trade-offs using a commercial HLS tool. We present quantitative performance and power analysis for LTE kernels. Through systematic design space exploration, we can achieve on average 20X performance speedup, or 7.6X energy saving compared to the baseline design.

1. Introduction

Recently, we have witnessed the explosive growth of Internet of Things (IoT). Collectively, the smart phones, 4G network hubs, data centers, etc. have created a super-connected IoT ecosystem. To support efficient communication among various IoT devices, the need for high bandwidth wireless network is increasingly urgent. 3GPP Long Term Evolution (LTE), which is a standard for 4G wireless communication, is designed to support implementation of the IoT infrastructure.

Designing IoT-friendly LTE chip is of paramount importance for the entire IoT ecosystem. However, modern IoT mobile devices have stringent power/energy requirements. Hence, the LTE chip should be power efficient. Nevertheless, previous approaches to implement LTE on CPU or GPU platforms [9], [10], [11] achieve very high performance improvement but incur very high power consumption, which makes them infeasible for low-power design requirement. Recently, FPGA have been employed as power efficient hardware accelerators for various applications [12]. In this work, we will explore the possibility of designing power-efficient LTE chips using FPGAs.

However, the programming challenge of using low-level RTL implementation makes FPGA development tedious, time-consuming, and difficult to debug. Fortunately, after decades of efforts in academia and industry, high level synthesis (HLS) tools have been widely adopted to alleviate the programming challenge of FPGA. In general, tens of magnitude of development efforts can be saved by employing HLS tools to automatically generate HDL code from the code written in high level language (C, C++, and SystemC) [5], [6], [7], [8]. The hardware accelerators generated by HLS can achieve tens or hundreds magnitude of performance improvement compared to the software implementations. Furthermore, by providing various optimization directives at high level, HLS tools can efficiently generate different hardware designs with different design trade-offs such as performance, power, resource utilization, and etc.

In this paper, we aim to quantitatively analyze the performance and power trade-offs of LTE on FPGAs using HLS tools. More specifically, we assemble an LTE benchmark suite, which is written in C++ and can be directly used as input for HLS tools. Then, we design an evaluation framework, which employs a commercial HLS tool to transform LTE software kernels written in C++ to hardware described in Verilog, and then implement these hardware kernels on FPGA by using a commercial logic synthesis tool. Finally, we provide quantitative performance and power analysis for different designs. The contributions of this paper are:

- We propose a framework to systematically generate the design of LTE kernels on FPGAs using HLS tools and explore the design space formed by different optimization options.
- We provide in-depth analysis of the trade-offs between performance and power for the pareto optimal points for LTE kernels.

Experiments show that through systematic design space exploration, we can achieve on average 20X performance speedup, or 7.6X energy saving compared to the baseline design.

2. Background

2.1 High level synthesis

High level synthesis is the process of automatically transforming the algorithms written in high level languages (e.g. C, C++, and SystemC) to RTL languages (e.g. Verilog, and VHDL). By using HLS tools, tens of magnitude of RTL design effort of traditional FPGA development could be saved [5], [6], [7], [8] which gives designers more flexibility to focus on the high level algorithm design instead of the tedious RTL implementation. State-of-the-art commercial HLS tools [8] incorporate a directive-guided compilation technique enabling the designers to directly add pragmas to algorithms written in high level language. These pragmas are then used to guide architecture-level optimizations (e.g. memory partition, loop unroll, pipeline, expression balancing, and etc.) during high level synthesis. Different combinations of pragmas will result in different performance and power trade-offs.

978-1-4799-8485-5/15/$31.00 ©2015 IEEE

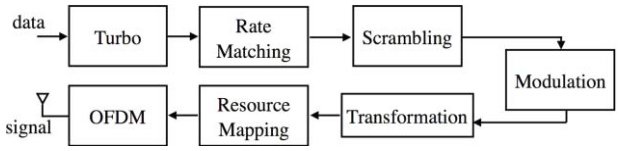

Figure 1. System flow graph of LTE uplink physical layer.

2.2 LTE workload

LTE, a standard of 4G wireless communication for mobile phones and data terminals, is designed to provide the backbone for the implementation of the IoT infrastructure. The system flow graph of LTE uplink physical layer is shown in Figure 1. LTE uplink physical layer consists of a series of kernels such as *Turbo*, *Rate Matching*, *Scrambling*, *Modulation*, and etc., and each kernel contains multiple key computation functions.

Designing IoT-friendly LTE chip plays an important role in the entire IoT ecosystem, and the power and performance are two major metrics for evaluating LTE chips in IoT devices. However, previous implementations of LTE focus on improving performance only [9], [10], [11], and it lacks of a systematic evaluation of different LTE designs with performance and power trade-offs.

Table 1. Architecture-level optimization.

Pragmas	Syntax
Pipeline	pipeline [options] <location>
Dataflow	dataflow [options]
Loop unroll	unroll [options] <location>
Array partition	array partition [options] <location> <array>

3. Evaluation Framework

3.1 Evaluation framework overview

The flow of our systematic framework is shown in Figure 2. First, we assemble an LTE benchmark suite written in C++, which could be directly used as input for HLS tools. Second, the software kernels and the corresponding optimization pragmas shown in Table 1 are both fed to Vivado HLS tool, and the C++ code will be automatically transformed to HDL (e.g. Verilog and VHDLs) code. Then, the functionality of LTE kernel written in HDL code is verified by C++/RTL co-simulation of Vivado HLS, and the latency in number of clock cycles is collected. The verified HDL code is then sent to Vivado tool to do logic and physical synthesis targeting at FPGAs. Lastly, the power consumption is recorded after placement and routing.

3.2 Architecture-level optimization

HLS tool provides various optimization options through pragmas for high performance and energy efficient designs. Different pragmas results in different performance, power, and resource utilization results. The pragmas used in our evaluation framework are shown in Table 1.

- *Pipeline*: A pipelined function or loop can process new input data set every *N* clock cycles, where *N* represents the initiation interval (*II*). The less the *II* is, the higher performance it will be achieved. But *pipeline* will also incur more power and resource utilization because of the increased execution parallelism.

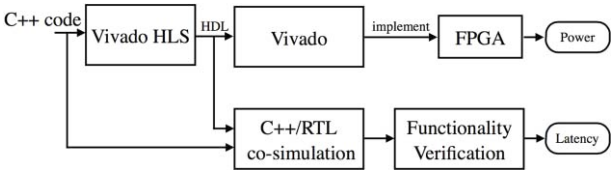

Figure 2. Performance and power evaluation framework

- *Dataflow*: Compared with *pipeline* optimization technique, *data flow* is to achieve parallelism at a coarse-grain level. By evaluating the interactions between different functions, *data flow* applies pipeline transformation to functions which can be executed in parallel.
- *Loop unroll*: When a loop is unrolled, multiple copies of a loop body will be created in hardware. Multiple copies of a loop work in parallel which increases the throughput and consumes more power and logic resources.
- *Array partition*: On-chip memory bandwidth sometimes may not be enough to support multiple read/write requests from parallel processing units at the same time. Array partition is thus utilized to partition a big memory block into multiple small blocks to increase the number of memory access ports as to support more execution units working in parallel.

3.3 Circuit-level optimization

Circuit-level optimization techniques such as *power gating*, *congestion alleviation*, *retiming*, and etc. are studied in previous studies. In this paper, we focus on exploring different target *maximum frequency*.

- *Maximum Frequency*: When the latency in number of clock cycles of a kernel is determined, the performance of this kernel is constrained by the *maximum frequency* the chip can achieve. However, in order to achieve higher frequency, less delay is allowed for each pipeline stage, which requires more flip-flop resources to support more pipeline stages. The increased pipeline stages and resource utilization will consume more power. Therefore, the *maximum frequency* is also an important factor that affects the performance and power trade-offs of LTE chip. In this work, we sweep five different *maximum frequency* values, which are 50, 67, 100, 125, and 200MHz, and the timing constraints are all satisfied after placement and routing.

Table 2. Pragmas and design exploration space for each kernel.

Kernel	Employed Pragmas	Design Space
Scrambling	*pipeline*, *unroll*	40
Modulation	*pipeline*, *unroll*, *array partition*	40
Rate Matching	*pipeline*, *unroll*, *dataflow*	40
Resource Mapping	*pipeline*, *partition*, *array partition*	40

4. Experiments

For each LTE kernel, we use Xilinx Vivado HLS tool to synthesize the code from C++ to Verilog and the target device is Xilinx VC707 development board. The optimization pragmas employed for each kernel are shown in Table 2. To verify the correctness of Verilog

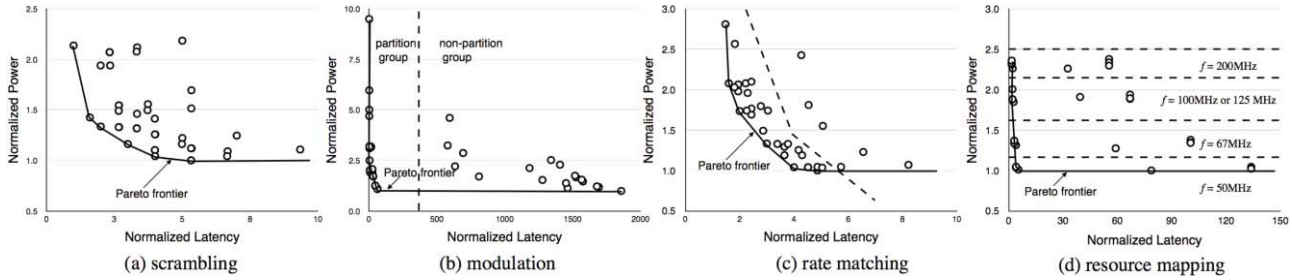

Figure 3. A set of plots characterizing the design space with different power and latency values (circles represents different design results, the solid line represents the Pareto frontier, and the power and latency are normalized to minimum power and latency respectively).

code, we employ C++/RTL co-simulation to verify the functionality of each kernel. Then we use Vivado to do logic synthesis, placement and routing. Clock period, power, and resource utilization data are obtained from post-placement and routing reports. The latency in clock cycles of each kernel is obtained from the reports of Vivado HLS. Note that since the input data size of each LTE kernel is fixed, the latency in cycles obtained from the Vivado HLS report is accurate. The hardware latency of each kernel in seconds is computed by multiplying the clock period and the measured clock cycles.

4.1 Performance and power trade-off analysis

After collecting power and performance results from our proposed evaluation framework, we plot the performance-power results in Figure 3 to further analyze the trade-offs between performance and power.

- *Scrambling*: The *Scrambling* kernel mainly consists of four loops, and each loop executes large amounts of floating-points calculations. Because of the inter-loop dependencies, the four loops must be executed one after another. For each loop, since there is no intra-loop dependency, *II=1* can be achieved if *pipeline* pragma is used, or the loop can be completely unrolled when *loop unroll* pragma is employed. Different pragma combinations between *pipeline* and *loop unroll* will result in different design trade-offs. Therefore, determining optimal pragma configuration is not trivial. As Figure 3 (a) shows, the performance-power points are evenly distributed inside the Pareto frontier. For the designs which are on Pareto frontier, the performance improvement can be up to 2.17X, or the energy be reduced by 4.84X compared to baseline design without optimization.
- *Modulation*: *Modulation* kernel has a complex structure, where six nested loops constitutes a three-level loop hierarchy. The loops in each loop level need to get values from corresponding 2-D data arrays which are implemented as on-chip memory in hardware. However, if we want to apply *pipeline* or *loop unroll* pragmas to a loop, the limited memory bandwidth becomes the bottleneck due to the increased number of parallel memory accesses. Therefore, we employ *array partition* to increase the memory bandwidth as to support more loops executing in parallel. As we can see from the Figure 3 (b), the points are categorized into two groups: (1) the partition group where *array partition* is used, and (2) the non-partition group where *array partition* is not applied. The reason for the better performance and power efficiency of the designs in the partition group is that the *array partition* largely increases the memory bandwidth thus allowing more execution units working in parallel. For the Pareto optimal designs, the performance can be improved by up to 1858X, or the energy be reduced by 174X.
- *Rate Matching*: The *Rate Matching* mainly consists of one nested loop to execute the key computation function, and two simple loops which just transfer the results to output data buffer. The performance of this kernel is mainly determined by the nested loop, and thus we focus on the nested loop in the following analysis. Because of the inter-loop dependencies, only the inner loop can be unrolled or pipelined. Therefore *pipeline* and *loop unroll* are applied to this loop. According to the results of Pareto optimal designs, the performance can be improved up to 8.22X, or the energy be reduced by 2.86X. We can further conclude from Figure 3 (c) that *Rate Matching* is a memory-bound kernel, because the designs on the left side of the dashed line with *pipeline* or *loop unroll* pragmas applied are concentrated around the Pareto frontier, resulting in Pareto optimal or near-optimal performance and power trade-offs. The designs that do not use *pipeline* or *loop unroll* pragmas yield unsatisfactory performance and power results.
- *Resource Mapping*: The *Resource Mapping* kernel has both intra- and inter- loop dependencies which make the potential optimization options limited. To be more specific, the computation function is mainly executed in two nested loops. Both loops have a three-level loop hierarchy, but only the most inner loop can be pipelined or unrolled due to the inter loop dependencies. To apply *pipeline* or *loop unroll* for the inner loop, the data array must be partitioned because of the intra-loop dependencies. According to Figure 3 (d), we can find that, the designs with both *pipeline* and *array partition* pragmas are Pareto optimal or near optimal, while the designs without *pipeline* or *array partition* pragmas yield far worse performance and power trade-offs. For the designs which are Pareto optimal, the performance can be improved up to 5.26X, or the energy be reduced by 1.39X. An interesting thing about this kernel is that most of the power consumption is from dynamic power, so the

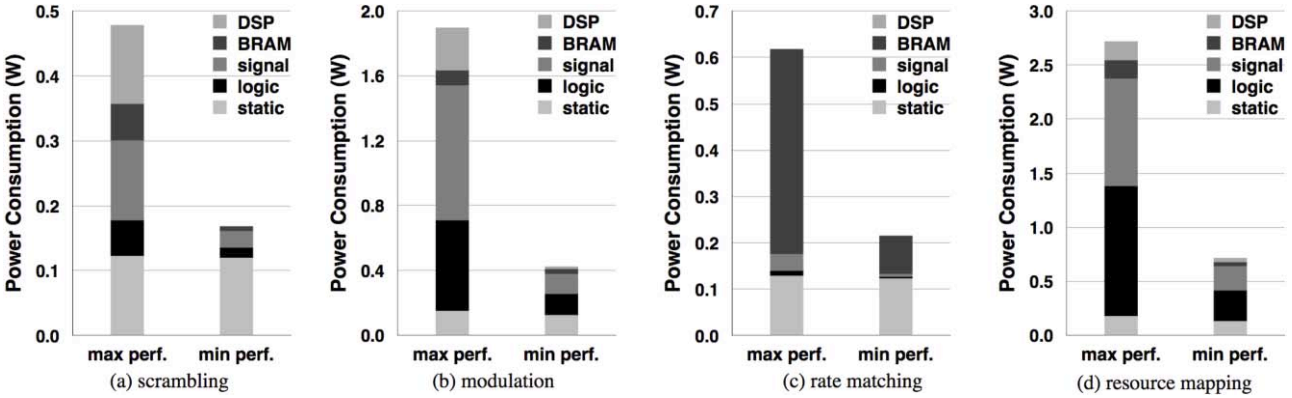

Figure 4. Power consumption breakdown for each LTE kernel (the min and max perf. refer to the designs which are Pareto optimal but with minimum and maximum performance respectively).

frequency affects the performance and power a lot. As Figure 3 (d) shows the power consumption increases with the increase of the operating frequency.

4.2 Power consumption breakdown

In this section, we further breakdown the power consumption of each LTE kernel. Two representative designs of each kernel are selected to be further analyzed: (1) the design which is Pareto optimal with the maximum performance, and (2) the design which is Pareto optimal but with the minimum performance.

The power consumption consists of static power and dynamic power. For static power, the designs with the maximum performance consume 17% more power than the designs with the minimum performance on average. The increase of static power is due to more utilized compute and memory resources which incur higher leakage power. In the contrast, the dynamic increases about 5 folds on average in designs with the maximum performance. There are two reasons for the dramatic increase of dynamic power. First, the employed optimization pragmas increase the activity factor resulting from more execution units working in parallel. For instance, the logic, signal, and DSP power increase drastically in *Scrambling*, *Modulation*, and *Resource Mapping* Kernels, and the BRAM power increases a lot in *Rate Matching* kernel. Secondly, the designs with maximum performance are operating in the maximum frequency at 200MHz, which further increases dynamic power consumption.

5. Conclusions

Nowadays, IoT-friendly LTE design requires the design generated by HLS to be both high performance and power efficient. Therefore, this study targets at analyzing the performance and power trade-offs by using HLS tools in the scenario of IoT-friendly LTE. We propose an efficient evaluation framework to analyze the performance and power trade-offs. Through design space exploration, we can achieve on average 20X performance speedup, or 7.6X energy saving compared to the baseline design.

6. References

[1] Q. Zheng, Y. Chen, R. Dreslinski, C. Chakrabarti, A. Anastasopoulos, S. Mahlke, and T. Mudge, "WiBench: An Open Source Kernel Suite for Benchmarking Wireless Systems," in *IISWC'13,* pp.123-132 (2013).

[2] Z. Zhang, et al., "AutoPilot: A Platform-Based ESL Synthesis System," in *High-Level Synthesis: From Algorithm to Digital Circuit*, 2008.

[3] Y. Sun, J. R. Cavallaro, and T. Ly, "Scalable and low power LDPC decoder design using high level algorithmic synthesis," in *SOCC,* pp.267-270 (2009).

[4] B. Reagen, Y. S. Shao, G. Y. Wei, D. Brooks, "Quantifying acceleration: power/performance trade-offs of application kernels in hardware," in *2013 ISLPED'13*, pp.395-400 (2013).

[5] W. Zuo, Y. Liang, P. Li, K. Rupnow, D. Chen, and Jason Cong. Improving high level synthesis optimization opportunity through polyhedral transformations. In *FPGA '13*, pp9-18 (2013).

[6] Y. Liang, K. Rupnow, Y. Li, D. Min, M. N. Do, and D. Chen, "High-level synthesis: productivity, performance, and software constraints", *JECE*, pp1-14 (2012).

[7] K. Rupnow, Y. Liang, Y. Li, D. Min, M. N. Do, and D. Chen, "High-level synthesis of Stereo Matching: Productivity, Performance, and Software Constraints", *FPT*, (2011).

[8] Z. Zhang, Y. Fan, W. Jiang, G. Han, C. Yang, and J. Cong, "AutoPilot: a platform-based ESL synthesis system," in *High-Level Synthesis: From Algorithm to Digital Circuit*, Springer (2008).

[9] K. Tan, J. S. Zhang, J. Fang, H. Liu, Y. S. Ye, S. Wang, Y. G. Zhang, H. T. Wu, W. Wang, and G. M. 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*, pp75-90 (2009).

[10] M. Wu, S. Gupta, Y. Sun, and J. R. Cavallaro, "A GPU Implementation of a Real-time MIMO Detector," in *IEEE Workshop on Signal Proceeding Systems on 2009*, pp303-308 (2009).

[11] Q. Zheng, Y. Chen, R. Dreslinski, C. Chakrabarti, A. Anastasopoulos, S. Mahlke, and T. Mudge, "Architecting an LTE Base Station with Graphics Processing Units," in 2013 *IEEE Workshop on Signal Processing Systems*, pp219-224 (2013).

[12] G. H. Wang, B. Yin, K. Amiri, Y. Sun, M. Wu, and J. R. Cavallaro, "FPGA Prototyping of a high data rate LTE Uplink Based Receiver," in *43rd Asilomar Conference on Signals, Systems, and Computers*, pp248-252 (2009).