ATHENa – Automated Tool for Hardware EvaluatioN: Toward Fair and Comprehensive Benchmarking of Cryptographic Hardware using FPGAs

Kris Gaj, Jens-Peter Kaps, Venkata Amirineni, Marcin Rogawski, Ekawat Homsirikamol, Benjamin Y. Brewster
ECE Department, George Mason University
4400 University Drive, Fairfax, VA 22030, USA
Email: {kgaj, jkaps, vamirin1, mrogawsk, ehomsiri, bbrewste}@gmu.edu

***Abstract*—A fair comparison of functionally equivalent digital system designs targeting FPGAs is a challenging and time consuming task. The results of the comparison depend on the inherent properties of competing algorithms, as well as on selected hardware architectures, implementation techniques, FPGA families, languages and tools. In this paper, we introduce an open-source environment, called ATHENa for fair, comprehensive, automated, and collaborative hardware benchmarking of algorithms belonging to the same class. As our first goal, we select the benchmarking of algorithms belonging to the area of cryptography. Algorithms from this area have been shown to achieve significant speed-ups and security gains compared to software when implemented in FPGAs. The capabilities of our environment are demonstrated using three examples: two different hardware architectures of the current cryptographic hash function standard, SHA-256, and one architecture of a candidate for the new standard, Fugue. All source codes, testbenches, and configuration files necessary to repeat experiments described in this paper are made available through the project web site.**

***Index Terms*—open-source; performance evaluation; benchmark tool;**

I. Introduction

The difficulties associated with a fair comparison of digital systems designed and modeled using hardware description languages, and implemented using FPGAs, can be divided into

- Evaluation Pitfalls: Mistakes that can be quite easily avoided if the person performing comparison is aware of potential dangers, and exercises appropriate caution and fairness; and
- General Objective Difficulties: Objective inherent difficulties that must be comprehensively addressed before a fair comparison is possible.

Examples of evaluation pitfalls include: Taking credit for improvements in technology, choosing a convenient (but not necessarily fair) performance measure, comparing designs with different functionality, comparing designs optimized using a different optimization target (speed, area, cost, power, balanced, etc.), comparing clock frequency after synthesis vs. clock frequency after placing and routing, etc. These mistakes can be most easily described using the phrase "comparing apples and oranges."

Objective difficulties are more challenging to overcome, and include lack of standard interfaces, influence of tools and their options, differences between a stand-alone performance vs. performance as a part of a bigger system, the dependence of the obtained results on the time spent for optimization, etc. [1].

Our project aims to address all aforementioned difficulties by developing an open-source benchmarking environment called ATHENa – Automated Tool for Hardware EvaluatioN [2]. The goal of our project is to spread knowledge and awareness about good performance evaluation practices (and this way eliminate or at least limit the evaluation pitfalls), and to develop the methodology and tools required to overcome objective difficulties.

The rest of the paper is organized as follows. In Section II, we discuss in detail the motivation for our project and its major goals. In Section III, we describe earlier work that inspired our research and development effort. Section IV gives an overview of our benchmarking environment, and describe its major features. Section V shows the benefits of our environment when applied to several case studies – the analysis and optimization of the hardware implementation of the cryptographic hash standard SHA-256, and four different types of comparisons targeting respectively two different algorithms, architectures, FPGA families, and versions of tools. We conclude the paper with the description of future work in Section VI, and the summary of major properties of our environment in Section VII.

II. Motivation and Goals

In almost every area of science and engineering, the same task can be realized using multiple competing algorithms. This statement is especially true in case of communications, digital signal processing, and cryptography. The choice of a particular algorithm depends strongly on its efficiency in software and hardware. One of the primary goals of our environment is to make the comparison of competing algorithms fairer and more comprehensive, especially for the case where reconfigurable hardware is a viable and advantages means of implementation. Although our environment can be used for comparison of

algorithms belonging to different fields, it is beneficial to focus first on algorithms belonging to one particular area: cryptography.

The reason why this area is appropriate include

- well documented speed-ups and security gains of FPGA implementations over software implementations,
- constantly evolving standards, due to the everlasting struggle between designers of new algorithms and cryptanalysts attempting to break them,
- strong need for fair evaluation associated with the way new cryptographic standards are being developed, namely through open competition of algorithms submitted by groups from all over the world.

Starting from the Advanced Encryption Standard (AES) contest organized by NIST in 1997-2000 [3], open contests have become a method of choice for selecting cryptographic standards in the U.S. and over the world. The AES contest in the U.S. was followed by the NESSIE competition in Europe [4], CRYPTREC in Japan, and eSTREAM in Europe [5].

Four typical criteria taken into account in the evaluation of candidates are: security, performance in software, performance in hardware, and flexibility. While security is commonly recognized as the most important evaluation criterion, it is also a measure that is most difficult to evaluate and quantify, especially during the relatively short period of time reserved for the majority of contests. The typical outcome is that, after eliminating a fraction of candidates based on security flaws, a significant number of remaining candidates do not demonstrate any easy to identify security weaknesses, and as a result are judged to have adequate security.

For example, during the AES contest, all five final candidates were identified by NIST, NSA, and cryptographic experts worldwide as possessing at least adequate security [3]. As a result, additional criteria were necessary to break the tie. Performance in software and hardware are next in line to clearly differentiate among the candidates for a cryptographic standard. Both criteria are very convenient – they are relatively easy to evaluate and quantify, objective, and of practical importance for the commercial viability (in terms of cost, speed, and energy consumption) of the end products incorporating the standard.

Interestingly, the differences among the cryptographic algorithms in terms of the hardware performance seem to be particularly large, and often serve as a tiebreaker when other criteria fail to identify a clear winner [3], [6].

At this point, the focus of attention of the entire cryptographic community is on the SHA-3 contest for a new hash function standard, organized by NIST [7]. The contest is now at its early stages, and the evaluation of candidates is scheduled to continue till the second quarter of 2012. Therefore, the development of our environment is perfectly aligned with the most important stages of the competition, when the results of the hardware performance comparison may have the highest possible impact.

Although facilitating a fair and comprehensive comparison among competing cryptographic algorithms is probably the easiest to identify goal of our project; this is not the only important goal we have in mind.

Progress in the art and science of digital system design requires the ability to fairly compare various ways of implementing the same algorithm. In hardware these different ways amount to different architectures (such as basic iterative, unrolled, pipelined, quasi-pipelined, etc.), different optimization tricks (such as precomputation, table look-up, etc.) and different ways of coding the same architecture using a hardware description language. Different implementation-level optimizations are presented at conferences and workshops, and it is common for their authors to compare their results with previous work. Unfortunately, the quality and fairness of these comparisons is often seriously flawed. Our environment is aimed at facilitating fair and comprehensive comparison of functionally equivalent architectures and implementations, and at exposing any evaluation pitfalls and unfair practices.

The third important goal of the performance evaluation is the identification of an implementation platform which is most suitable for a specific design of a given algorithm. Factors to be taken into account include speed, cost, power and energy consumption, physical dimensions, etc. In the most general case, the designer must first choose between three major semiconductor technologies: microprocessors (including microcontrollers and DSPs), FPGAs, and ASICs. However, in the most common scenario, the semiconductor technology is predetermined by other factors, such as production volume, non-recurring costs, physical size, security requirements, etc., and the remaining choice concerns the particular device within a given class. In particular, in the case of FPGAs, the choice concerns a preferred vendor, family, and device within a family. This choice can be significantly facilitated by the use of our environment.

Finally, the obtained results may be a strong function of hardware description languages, tools, and tool versions. Benchmarking such tools and languages is a fourth important goal of our project. A comprehensive evaluation of equivalent results obtained using different tools and languages for a wide class of algorithms, such as cryptographic algorithms, will be of great help for both hardware designers and tool developers.

In summary, our goal is to develop a methodology and a computer environment that would allow for the *comprehensive*, *fair*, *reliable* and *practical* software and hardware performance comparison among various

- algorithms,
- implementation methods,
- platforms,
- languages and tools.

III. Previous Work

FPGA vendors by themselves have recently started the development of tools for the exploration of implementation options. A good example is ExploreAhead [8] from Xilinx, which is a part of the high-level optimization tool called PlanAhead. Similarly to ATHENa, ExploreAhead allows executing multiple implementation runs based on user defined

strategies or predefined strategies shipped together with the tool. Each strategy corresponds to a certain set of options of the Xilinx mapping, placing and routing tools. Based on these strategies, a user can execute multiple implementation runs, each corresponding to a different optimization strategy. These runs can be parallelized to take advantage of multi-core CPU machines.

Compared to ExploreAhead, which focuses exclusively on Xilinx devices, ATHENa is intended to provide similar capabilities for designers and scientists interested in exploring FPGA devices from several vendors. In terms of optimization, ATHENa is aimed at achieving the best possible performance, rather than a target performance, defined by any actual system specification. Additionally, the optimization strategies developed within ATHENa will be more closely related to a particular class of digital systems, starting from (but certainly not limited to) the cryptographic hash functions, selected as our immediate exploration target because of the on-going SHA-3 competition [7].

In the specific area of performance evaluation of cryptographic algorithms, our inspiration comes from the eBACS project, started by Daniel J. Bernstein and Tanja Lange in 2006 [9]. Within this project, a special tool called SUPERCOP was developed in order to facilitate comparison of *software* implementations of cryptographic algorithms. This open-source tool supports the choice of best compilation options from among over 1200 different combinations. It also allows the actual execution time measurements to be performed on multiple computer systems of various kinds. The project supports multiple classes of cryptographic algorithms (such as secret key block ciphers, stream ciphers, hash functions, etc.), and for each of them defines a standardized Application Programming Interface (API) (an equivalent of the hardware interface in digital system design). The eBACS project calls for and facilitates the separation of designers of cryptographic algorithms from evaluators responsible for their benchmarking. We believe that in spite of clear and significant differences between software and hardware benchmarking (such as compilation/implementation time, ways of determining the execution time, management of memory hierarchy, etc.), the major ideas and benefits of the eBACS project can be applied to the realm of FPGAs.

IV. Environment

A. Overview

We have developed a prototype of ATHENa: Automated Tool for Hardware EvaluatioN [2]. At the heart of our tool is a set of scripts written in Perl aimed at an *automated* generation of *optimized* results for *multiple* hardware platforms.

The only software required to run the tool is an interpreter of Perl, which is available for free. The tool also assumes that FPGA design environments are already installed on the system executing the scripts. The users can use either free, educational, or commercial versions of these FPGA design environments.

The general idea of our hardware evaluation environment is shown in Fig. 1.

The ATHENA Server is a focal point of the environment. It hosts the project web site [2], and repository of project scripts and sample configuration files. In the near future, this server is intended to host a large database of results. Each algorithm will be initially represented in the project database by several entries, including algorithm specification (e.g., Federal Information Processing Standard, FIPS) reference implementation in C (or other programming language), and test vectors. In the next step, we will develop and store for each of these algorithms one or more proposed standard hardware interfaces, and the corresponding testbenches.

A hardware designer can download the aforementioned entries to his local machine, and use them to develop his/her implementation of a given algorithm in the form of Hardware Description Language (HDL) code. The designer can also choose his own interface and develop the corresponding testbench by himself. In this case, the initial download of information from the server is not necessary. After the HDL code is ready, and its functionality verified through simulation, the actual performance evaluation process can begin.

At this point, the user downloads our scripts and sample configuration files to his local machine. He/she modifies configuration files, so they contain proper information about the location of HDL source files, location of tools, target hardware platforms (e.g. Xilinx Virtex 5 and Altera Cyclone III), and other parameters required by the scripts. The user then starts the scripts that run the FPGA implementation in the batch mode, and generate the result summary in the form of text files suitable for the designer's review.

In the near future, our environment will be extended with the database of results. The ATHENa scripts will generate the necessary database entries automatically. The designer will be in position to first review the human-friendly result summary, and only afterwards to decide whether to submit

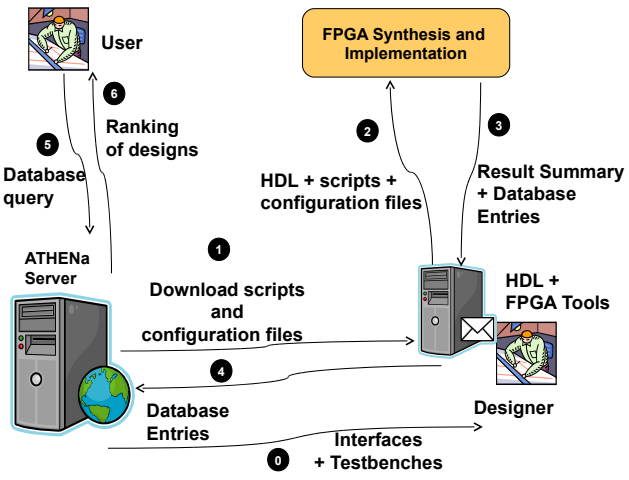

Fig. 1: Data flow within the hardware evaluation environment

the corresponding database entries to the project database.

The important feature of our approach is that all computations are performed on a local machine of the designer, and thus the HDL code never leaves this machine, and is never a subject to interception by any third party, including the project server administrators.

On the other hand, the user must have all FPGA tools and libraries necessary for the evaluation installed on his/her own machine.

B. Features

The main features of our environment include:

1) Running all steps of synthesis, implementation, and timing analysis in batch mode: This is a very important property, as it allows running time-consuming optimizations, without any user supervision, over long periods of time, such as nights, days, or even weeks.

2) Support for devices and tools of two major FPGA vendors: Xilinx and Altera: Xilinx and Altera account for about 90% of the FPGA market. Their FPGA devices differ considerably in terms of the structure of a basic building block: configurable logic block (CLB) for Xilinx, and logic element (LE) for Altera. They also differ in terms of dedicated hardwired units, such as blocks of memory, multipliers, DSP units, etc. As a result, the ranking of algorithms or architectures obtained using devices of one FPGA vendor may not carry to the devices of another vendor.

3) Generation of results for multiple FPGA families of a given vendor, e.g. Xilinx: Spartan 3, Virtex 5; Altera: Cyclone III, Aria II, Stratix IV: Our tool allows specifying as target platforms multiple families of FPGA devices of each of the two major vendors.

Every vendor supports over time two or three classes of families, which are optimized respectively for performance, cost and power consumption, and performance to cost ratio. Families belonging to different classes differ significantly, and therefore may produce substantially different results and rankings. Families belonging to the same class also gradually evolve over time. Our tool allows an easy and comprehensive investigation of the dependence of results and rankings on the FPGA families.

4) Automated choice of a device within a given family of FPGAs assuming that the resource utilization does not exceed a certain limit, e.g. 80% of CLB slices or 50% of BRAMs: A maximum clock frequency of a circuit implemented using an FPGA is a function of device resource utilization. When the device utilization reaches 80%–100% in terms of one of the critical resources, such as configurable logic blocks or Block RAMs, the performance degrades. This effect is caused mostly by the difficulties associated with routing in congested circuits. The utilization threshold at which the performance degradation begins is a function of an FPGA family and the implemented circuit. ATHENa supports first determining these thresholds separately for each family of FPGAs and each class of digital circuits. Our environment includes special library files characterizing all devices of a given FPGA family in terms of available resources. The tool is then able to match information from these library files, with the maximum percentage of resources permitted to be used without performance degradation, and select an FPGA device within a given family automatically.

5) Automated optimization of results aimed at one of the three optimization criteria: speed, area, and ratio speed to area: Results generated by the FPGA tools depend highly on the choice of multiple options and the contents of constraint files. Variation of results obtained by changing just a single option may easily exceed 25%.

At this point, ATHENa contains two design space exploration functions: Placement Search and Exhaustive Search.

Placement Search permits the exploration of result dependencies on the starting point of placement. This starting point is determined by the options of the FPGA implementation tools called: Cost Table in Xilinx tools, and Seed in Altera tools. Cost Table can take any integer value between 1 and 100, and Seed any value between 1 and 2^{32}. Both parameters are by default set to 1. Exploring the full range of these parameters may be computationally prohibitive, especially in case of Altera, so a representative subset of the full range needs to be selected.

Exhaustive Search is a superset of Placement Search and extends the set of options to be explored by other options, such as: optimization target (area, speed, or balanced), optimization level, maximum fanout, multiple target clock frequencies, etc. All options are divided into two levels. Level 1 options are changed first, while keeping Level 2 options at their default values. Afterwards, two (or more) sets of Level 1 options are selected and kept constant while Level 2 options are explored.

6) Automated verification of a design through functional simulation, run in batch mode: Our tool has an additional capability of simulating designs in the batch mode in order to verify their correct functionality. The verification is based on a testbench utilizing test vectors stored in a file, and providing a binary answer whether the circuit operates correctly or not.

Sample testbenches and hardware interfaces will be provided for the most common cryptographic algorithms (including all NIST standards). One such testbench has already been published at the ATHENa web site. This testbench can be used for the verification of implementations of 14 round-two candidates for the new SHA-3 standard, as well as implementations of current standards SHA-1 and SHA-2.

Designers themselves will be responsible for designing testbenches for any new algorithms, based on generic template files and coding guidelines made available through the project web site. The advantage of simulation in batch mode is that it can be run without any supervision for a long time.

V. Case Studies

For our case studies, illustrating characteristic features and capabilities of ATHENa, we have selected two implementations of the current cryptographic hash function standard, SHA-256, and one implementation of an alternative algorithm, called Fugue-256, competing in the contest for the new hash

function standard SHA-3 [7]. SHA-256 has been developed by NSA, and it was standardized by NIST in 2002 [10], Fugue was developed by IBM in 2008-2009, in response to the NIST call for SHA-3 candidates. Out of several hardware architectures of SHA-256, we have selected architectures referred to as basic loop and architecture with rescheduling. The former is the most straightforward sequential implementation of the algorithm, the latter is an optimized architecture, developed by Chaves et al. [11], optimized for the maximum throughput to area ratio.

Efficient implementations of all three designs have been developed by our group in VHDL. These implementations follow a generic interface suitable for the majority of modern cryptographic hash functions, including SHA-1, SHA-2, and SHA-3 candidates. The implementations were verified using a generic testbench, in which only an external test vector file is specific to a given hash function algorithm. The synthesizable source codes, the testbench, and the specification of the generic interface are all available at the ATHENa project web site [2].

Our first case study aims at developing a heuristic optimization strategy offering an acceptable trade-off between time spent on optimization, and the quality of the obtained results. This study was performed independently for each of the three described above designs. Below, we present the results for a single selected design: SHA-256 in the architecture with rescheduling. The results obtained for the remaining two designs were quite comparable.

In order to optimize the choice of an FPGA device within a given family, we have first determined the dependence of the maximum clock frequency on the CLB slice utilization. In order to do that, we have built a parameterized circuit comprised of a cascade of N SHA-256 units, separated by registers. We have then selected a Spartan 3 device, xc3s4000fg1156-5, for which one unit of SHA-256 takes about 3.33% of CLB slices. This way by changing parameter N, we are able to determine the maximum clock frequency of our circuit for the CLB slice utilization ranging from 3.33% to 96.67%. All clock frequencies have been obtained using Exhaustive Search with 48 sets of Level 1 parameters described below. Based on the dependence shown in Fig. 2, we have selected a threshold of 80%, as a value beyond which the maximum clock frequency deteriorates by a factor larger than 10%.

In the next step, we have run ATHENa in the single_run best_match mode with the value of the parameter MAX_SLICE_UTILIZATION set to 80%. In the result, the smallest Spartan 3 device, for which the CLB slice utilization does not exceed 80% was determined to be xc3s200ft256-5.

In order to optimize the circuit for the maximum throughput to area ratio, the Exhaustive Search function of ATHENa was employed. The following parameters have been changed in Phase 1 of Exhaustive Search:

- optimization target for synthesis: area, speed
- maximum fanout: 50, 100, 500
- optimization target for mapping: area, speed
- optimization effort level for mapping: medium, high
- optimization effort level for placing and routing: medium, high.

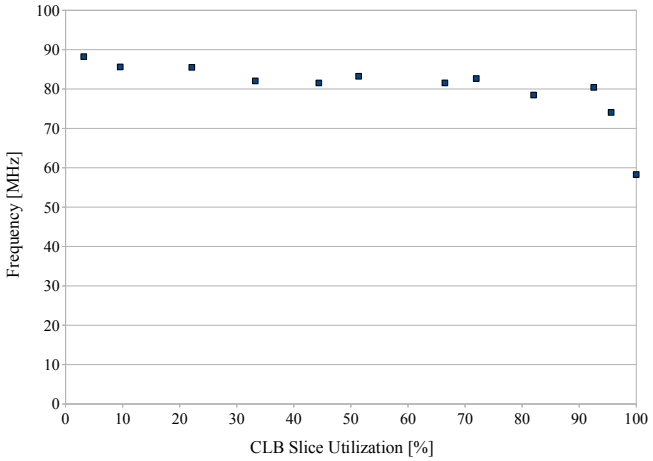

Fig. 2: Dependence of the maximum clock frequency on the CLB slice utilization.

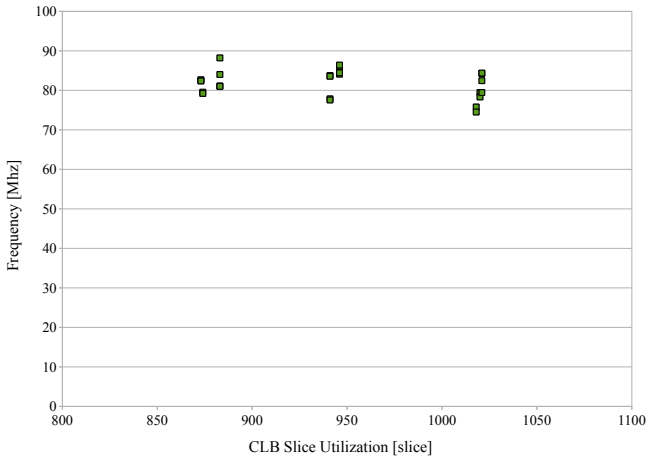

Fig. 3: Results of the Exhaustive Search for 48 sets of Level 1 parameters.

The total number of parameter sets tested was $2^4 * 3 = 48$. The target clock frequency was set to default. The results of this parameter space exploration are shown in Fig. 3. Out of 48 parameter sets, we have chosen one, with the best ratio of the maximum clock frequency to the CLB slice utilization for further processing. This set corresponds to the: optimization target for synthesis = area, maximum fanout = 100, optimization target for mapping = area, optimization effort level for mapping = medium, and optimization effort level for placing and routing = medium. The total execution time of this phase was equal to about 1.5 hr on the 2.66 GHz Intel Core 2 Duo VPro.

For this set of Level 1 options, we have run the synthesis and implementation with 100 different values of the parameter Cost Table, determining the starting point of placement. In Fig. 4, we show the distribution of the maximum clock frequencies obtained using these 100 values of Cost Table. Each bar in the diagram represents the number of Cost Table values, for which the maximum clock frequency falls within a given 1 MHz range. A black mark on the bar, represents the default

Cost Table value equal to 1. The grey marks on the bars represent the number of Cost Table values from the reduced set $\{21, 41, 61, 81\}$ falling within the same range. Together with the black bar, these bars represent a reduced-time exhaustive search taking only 5% of time used for the full-time exhaustive search.

In Figs. 5, 6, 7 and 8, we demonstrate that the obtained dependencies are a strong function of the target clock frequency. In particular, when the target clock frequency is either set to default, or is much higher than the achievable clock frequency, the spread of the actual clock frequencies is quite large. Requesting a target clock frequency that is realistic causes that the spread becomes narrower, as shown in Fig. 6. When the target clock frequency is smaller than the frequency that can be easily achieved by the tools, the distribution becomes very narrow, and the actual clock frequency only marginally exceeds the target value (see Fig. 5). In all the aforementioned diagrams, green bars denote clock frequencies higher than the target clock frequency, and red bars denote frequencies lower than the target clock frequency.

The best actual clock frequencies were achieved for the case of the target clock frequency equal to 90 MHz, as shown in Fig. 7. In this case, the maximum clock frequencies found using full-time exhaustive search, reduced-time exhaustive search, and single_run, were equal respectively to: 90 MHz, 88 MHz, and 83 MHz. Thus, the reduced-time exhaustive search gives results falling within approximately 2% from the best value obtained using full-time search, and it outperforms the single_run by 5 MHz (approximately 6%).

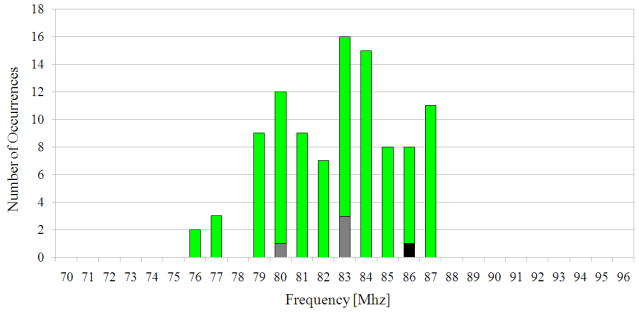

Fig. 4: Distribution of the actual clock frequencies for the default target clock frequency with 100 values of the cost table.

Overall, the obtained improvement of the maximum clock frequency compared to the Single Run with the default values of all parameters, (including Level 1 parameters) was equal to 12.5% (from 80 MHz to 90 MHz) for the full-time exhaustive search (taking about 5 hrs on the 2.66 GHz Intel Core 2 Duo VPro), and 10% (from 80 MHz to 88 MHz) for the reduced-time exhaustive search (taking about 2 hrs). This improvement is a strong function of an FPGA family and a particular circuit.

In general, our experiments demonstrated that the Exhaustive Search of ATHENa is a viable option for improving the implementation results at least for medium size circuits. The execution time of this search can be substantially reduced, using heuristic algorithms, at the cost of only minor degradation in the values of optimized results.

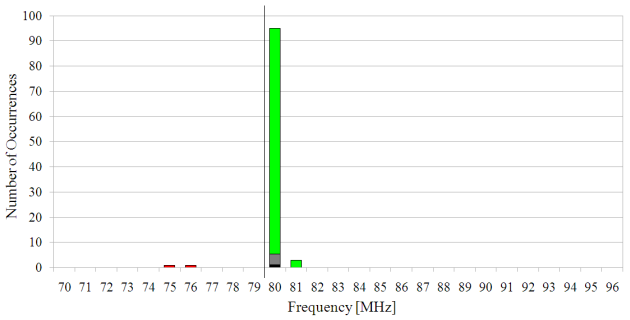

Fig. 5: Distribution of the actual clock frequencies for the target clock frequency equal to 80 MHz.

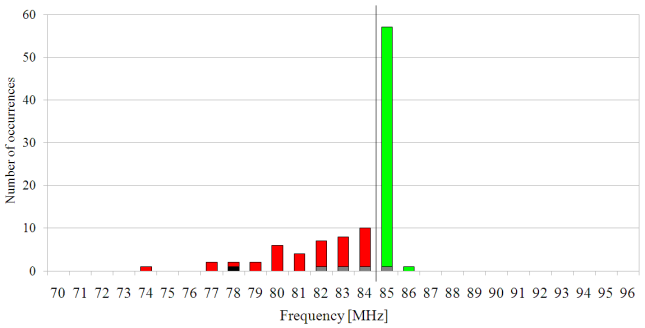

Fig. 6: Distribution of the actual clock frequencies for the target clock frequency equal to 85 MHz.

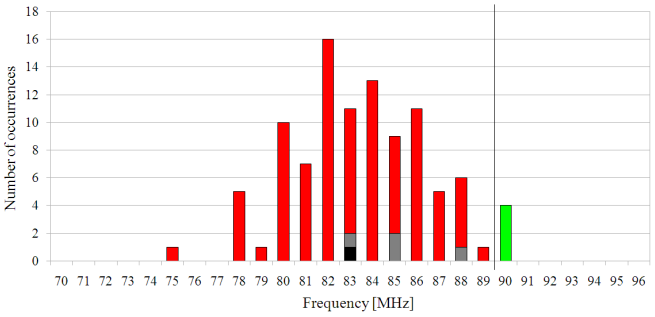

Fig. 7: Distribution of the actual clock frequencies for the target clock frequency equal to 90 MHz.

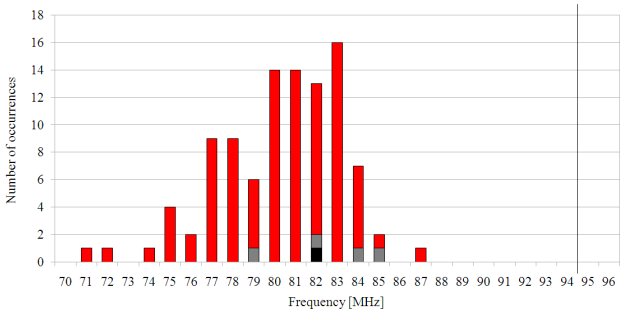

Fig. 8: Distribution of the actual clock frequencies for the target clock frequency equal to 95 MHz.

The similar experiment was repeated for Altera Cyclone II.

The most important finding was that the results after placing and routing were a very weak function of the requested implementation frequency. As a result, we have decided to follow different heuristic optimization strategies for FPGA devices from Xilinx and Altera.

For Xilinx FPGAs, we first search for the best target clock frequency. This search involves several single runs of tools, with the target clock frequency first set to the default value, and then gradually changed using the binary search algorithm, based on the corresponding actual clock frequency obtained from a given run. For the best target clock frequency obtained this way, we run exhaustive search, with the number of option sets reduced from 48 to 8 compared to our original experiment. Finally, for the best set of options returned by exhaustive search, we run placement search, with the number of initial placement positions reduced from 100 to 5 compared to the initial experiment. The total number of runs required by this strategy is in the range from 15 to 20.

For Altera, we apply directly exhaustive search, with the number of tested option sets equal to 12. We follow with the placement search, with 5 initial placement positions. The total number of runs is thus always equal to 17.

We then apply these heuristic optimization strategies to four different types of evaluations described earlier in Section II. In these evaluations, we compare respectively algorithms (SHA-256 vs. Fugue-256), architectures (basic loop vs. rescheduling), FPGA families from various vendors (Xilinx Spartan 3 vs. Altera Cyclone II), and tool versions (Xilinx ISE v. 9.1 vs. v. 11.1). The results of these comparisons are summarized in Tables I–IV. In each table we present first the results after single run of the tools (column "Single") and then results after optimization (column "Opt.). We also calculate the ratio of each result after optimization to the result before optimization (column "Ratio"). The last parameter listed in each table is the Optimization Time (denoted by "Opt. Time") given in minutes. It should be noted that this optimization time is typically smaller than one hour, which is typically a small fraction of the total development time.

From Table I, we can see that Fugue outperforms SHA-256 in terms of throughput, but is inferior in terms of area and the throughput to area ratio. Additionally, the optimization of SHA-256 improves area and throughput almost equally, while in Fugue, it affects practically only throughput. From Table II, after optimization, the architecture with rescheduling outperforms basic loop in terms of all performance measures. From Table III, Cyclone II outperforms Spartan 3 in terms of throughput and the throughput to area ratio, with the difference between both FPGA families decreasing after optimization. Finally, based on Table IV, somewhat surprisingly, the newer versions of tools give worse throughput and worse area after optimization. At the same time, they offer slightly better or comparable results after a single run.

All four tables demonstrate a potential for generating interesting, non-trivial, and sometimes unexpected results regarding the properties of various algorithms, architectures, FPGA families, and FPGA tools.

TABLE I: Comparison of two cryptographic hash function algorithms: SHA-256 and Fugue-256 using Xilinx Spartan 3

	SHA-256			Fugue-256		
	Single	Opt.	Ratio	Single	Opt.	Ratio
Frequency [MHz]	79.46	88.22	1.11	34.38	40.10	1.17
Area [CLB slices]	1020	883	0.87	3987	3873	0.97
Throughput [Mbit/s]	625.9	694.9	1.11	1100.2	1283.2	1.17
Throughput/Area	0.61	0.79	1.30	0.28	0.33	1.18
Opt. Time [min]	2.15	42.30	18.89	5.16	105.23	20.08

TABLE II: Comparison of two different hardware architectures of SHA-256 using Altera Cyclone II

	Basic Loop			Rescheduling		
	Single	Opt.	Ratio	Single	Opt.	Ratio
Frequency [MHz]	106.47	108.49	1.02	105.50	110.69	1.05
Area [LE]	2291	2216	0.97	2019	2015	1.00
Throughput [Mbit/s]	838.7	854.6	1.02	831.0	871.8	1.05
Throughput/Area	0.366	0.386	1.05	0.412	0.433	1.05
Opt. Time [min]	0.42	13.02	18.61	0.41	12.58	19.07

VI. Future Work

A. New Features

Several new features of our environment are currently under active development, and are likely to become available during 2010. The release schedule can be found on the Athena webpage [2]. These features include:

1) Additional FPGA vendors: In the near future our environment will be extended to support other FPGA vendors, such as Actel and Lattice Semiconductor.

2) Support for Windows and Linux: The majority of FPGA design environments (including those from Xilinx and Altera) operate under both Windows and Linux. After the initial development of our tool under Windows, its operation will be extended into Linux.

3) Graphical User Interface (GUI): In the current version of the ATHENa environment, the preparation of each evaluation run is done by editing sample configuration files using an arbitrary text editor. In the second phase, a GUI tool will be developed to facilitate the preparation of configuration files, and display of generated results.

TABLE III: Comparison of two different target hardware platforms: Xilinx Spartan 3 and Altera Cyclone II for SHA-256 (architecture with rescheduling). Area for Xilinx Spartan 3 is given in Logic Cells (LC), which are a half of a CLB slice, in order to make this parameter comparable to area for Altera expressed in Logic Elements (LE).

	Xilinx Spartan 3			Altera Cyclone II		
	Single	Opt.	Ratio	Single	Opt.	Ratio
Frequency [MHz]	79.46	88.22	1.11	105.50	110.64	1.05
Area [LC or LE]	2040	1776	0.87	2019	2015	1.00
Throughput [Mbit/s]	625.9	694.9	1.11	831.0	871.8	1.05
Throughput/Area	0.312	0.391	1.28	0.412	0.433	1.05
Opt. Time [min]	2.15	42.30	18.89	0.51	14.20	17.27

TABLE IV: Comparison of two different versions of tools: Xilinx ISE Design Suite v.11.1 vs. v. 9.1 for SHA-256 (architecture with rescheduling)

	Xilinx ISE v. 9.1			Xilinx ISE v. 11.1		
	Single	Opt.	Ratio	Single	Opt.	Ratio
Frequency [MHz]	77.87	92.58	1.19	79.46	88.22	1.11
Area [CLB Slices]	1020	873	1.17	1020	883	0.87
Throughput [Mbit/s]	613.4	729.2	1.19	625.9	694.9	1.11
Throughput/Area	0.601	0.835	1.39	0.614	0.787	1.28
Opt. Time [min]	2.17	42.20	18.24	2.15	42.30	18.89

4) Adapting to Other Domains: Additionally, ATHENa can be easily applied to domains different than cryptography, such as digital signal processing or communications. In such case, new heuristic optimization algorithms may need to be developed to better match features of these new classes of applications. In the longer term, our environment can be extended to cover ASICs (Application Specific Integrated Circuits).

VII. Conclusions

We have proposed and substantially advanced the development of an open-source tool, called ATHENa, for a fair, comprehensive, reliable, and practical benchmarking of digital systems using FPGAs from various vendors.

The most important features characterizing our environment are as follows:

- *Comprehensive*: The environment supports evaluation using multiple FPGA devices from several vendors.
- *Automated*: All tools run in batch mode, without the need for any user supervision.
- *Collaborative*: The environment allows and facilitates benchmarking by hundreds of designers from all over the world. As a result the effort on development, debugging, and optimization of codes is shared by a large number of designers, each of which can specialize in a single type of implementation platform and a single set of tools.
- *Practical*: Our environment supports but does *not* require revealing the source codes; as a result it can be safely used by a wide range of designers from academia, industry, and government unable to place their codes in public domain because of intellectual property or export restrictions issues.
- *Distributed*: The majority of the most time consuming computations (including all phases of hardware design and optimization) are performed on local machines of individual designers using tools they already have licenses for, and are familiar with.
- *Optimized*: Our scripts will make the best effort to select the best options of tools used for synthesis and implementation in FPGAs. In order to create such scripts, a comprehensive set of computationally intensive experiments will be performed during this project in order to select the best optimization strategy for each available tool and implementation platform.
- *With single point of contact*: Our project server will work as a single point of contact, and will contain all information necessary to perform benchmarking, and to share, look up, and compare the results.

The first big test of our environment will be its application to the evaluation of candidates submitted to the SHA-3 contest for a new hash function standard, organized and co-ordinated by NIST. At the time of writing, 14 candidates remain in the competition.

The environment will continue to serve the cryptographic and FPGA community for years to come, providing comprehensive and easy to locate results for multiple cryptographic standards and other classes of algorithms. Researchers all over the world will benefit from the capability of fairly, comprehensively, and automatically comparing their new algorithms, hardware architectures, and optimization methods against any previously reported work. The designers will benefit from the capability of comparing results of implementing the same algorithm using multiple FPGAs from several major vendors, and will be able to make an informed decision about the choice of the implementation platform most suitable for their particular application. Finally, the developers and users of tools will benefit from the comprehensive comparison done across tools from various vendors, and from the optimization methodologies developed and comprehensively tested as a part of this project.

References

[1] S. Drimer, "Security for volatile FPGAs," Chapter 5: The meaning and reproducibility of FPGA results, Ph.D. Dissertation, University of Cambridge, Computer Laboratory, Nov 2009, uCAM-CL-TR-763.

[2] "ATHENa Project Website," http://cryptography.gmu.edu/athena/.

[3] J. Nechvatal *et al.*, "Report on the development of the Advanced Encryption Standard (AES)," Oct. 2000, http://csrc.nist.gov/archive/aes/round2/r2report.pdf.

[4] B. Preneel *et al.*, "Final report of European project number IST-1999-12324, named New European Schemes for Signatures, Integrity, and Encryption," Apr. 2004, https://www.cosic.esat.kuleuven.be/nessie/Bookv015.pdf.

[5] M. Robshaw and O. Billet, *New Stream Cipher Designs: The eSTREAM Finalists*. Springer, 2008.

[6] K. Gaj and P. Chodowiec, "Fast implementation and fair comparison of the final candidates for Advanced Encryption Standard using Field Programmable Gate Arrays," *LNCS 2020, Progress in Cryptology - CT-RSA 2001, Ed. D. Naccache, RSA Conference 2001 - Cryptographers' Track*, pp. 84–99, Apr. 2001.

[7] "Cryptographic hash algorithm competition," http://csrc.nist.gov/groups/ST/hash/sha-3/.

[8] M. Goosman, R. Shortt, D. Knol, and B. Jackson, "ExploreAhead extends the PlanAhead performance advantage," *Xcell Journal*, pp. pp. 62–64, Third Quarter 2006.

[9] "eBACS: ECRYPT Benchmarking of Cryptographic Systems," http://bench.cr.yp.to.

[10] *Secure Hash Standard (SHS)*, National Institute of Standards and Technology (NIST), FIPS Publication 180-2, Aug 2002, http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf.

[11] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis, "Cost-efficient SHA hardware accelerators," in *IEEE Transactions on Very Large Scale Integration Systems*, Aug 2008, pp. 999–1008.