ATHENA - AUTOMATED TOOL FOR HARDWARE EVALUATION: SOFTWARE ENVIRONMENT FOR FAIR AND COMPREHENSIVE PERFORMANCE EVALUATION OF CRYPTOGRAPHIC HARDWARE USING FPGAS

by

Venkata Amirineni

A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Engineering

Committee:

KGaj	Dr. Kris Gaj, Thesis Director
[illegible]	Dr. William Sutton, Committee Member
[illegible]	Dr. Jens-Peter Kaps, Committee Member
[illegible]	Dr. Andre Manitius, Department Chairman
[illegible]	Dr. Lloyd J. Griffiths, Dean, The Volgenau School of Information Technology and Engineering
Date: 7/30/2010	Summer Semester 2010 George Mason University Fairfax, VA

ATHENa - Automated Tool for Hardware EvaluatioN: Software Environment for Fair and Comprehensive Performance Evaluation of Cryptographic Hardware using FPGAs

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Venkata Amirineni
Bachelor of Science
George Mason University, 2008

Director: Dr. Kris Gaj, Associate Professor
Department of Electrical and Computer Engineering

Summer Semester 2010
George Mason University
Fairfax, VA

Dedication

I dedicate this thesis to my family and friends. Especially to my parents Venkata Narayana and Vedavathi Amirineni.

Acknowledgments

I would like to thank several people for helping me through the various stages of my thesis. First of all, is my avdisor Dr. Kris Gaj, who went out of his way to help me every possible way he could. Thank you Dr. Gaj. Next, I want to thank Dr. Jens-Peter Kaps, Marcin Rogawski, Ekawat Homsirikamol and Rajesh Velegalati for helping me with the design and implementation of ATHENa. Without the help of these invidicuals, ATHENa development would have been tedious.
I also want to extend my apperitiation to the following individuals:

- All the members of CERG group, for providing a friendly and stimulating environemnt to work in.
- The students of ECE 545 and ECE 645 classes of Fall 2009 and Spring 2010, for testing and providing valuable feedback for ATHENa improvement.

Table of Contents

List of Tables

List of Figures

Abstract

ATHENA - AUTOMATED TOOL FOR HARDWARE EVALUATION:
SOFTWARE ENVIRONMENT FOR FAIR AND COMPREHENSIVE PERFORMANCE EVALUATION OF CRYPTOGRAPHIC HARDWARE USING FPGAS

Venkata Amirineni, MS

George Mason University, 2010

Thesis Director: Dr. Kris Gaj

Fair comparison of the hardware efficiency of cryptographic algorithms, modeled in Hardware Description Languages and implemented using FPGAs, is a complex 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.

The development of new cryptographic standards through contests, such as AES, eSTREAM, and SHA-3 competitions, requires fair comparison of multiple cryptographic algorithms in terms of their hardware efficiency. To address this issue and to provide a comprehensive environment for an efficient evaluation of multiple algorithms, ATHENa, Automated Tool for Hardware EvaluatioN, has been developed.

ATHENa facilitates fair, comprehensive, reliable, and automated comparison of cryptographic algorithms, hardware architectures, FPGA families, as well as FPGA tools and HDL languages. In this seminar, we present the common pitfalls and difficulties involved with the fair comparisons and demonstrate the capabilities of ATHENa through several case studies.

Chapter 1: Introduction

1.1 Statement of the Problem

Fair comparison of hardware implementations of cryptography is at the basis of progress in cryptographic engineering. New hardware architectures and optimization techniques have to be compared to the current state of the art. The development of new cryptographic standards through contests, such as AES [1], eSTREAM [2], and SHA-3 competitions [3], requires fair comparison of multiple cryptographic algorithms in terms of their hardware efficiency.

Revealing source codes is often not an option because of the authors' concerns regarding their intellectual property rights, export restrictions, and/or possible loss of profits from licensing hardware cores.

This thesis reports on a comprehensive environment, Automated Tool for Hardware EvaluatioN (ATHENa) [4] that facilitates the fair comparison of cryptographic implementations, modeled in Hardware Description Languages (HDLs), without revealing the source codes.

The project was inspired by a similar environment for comparing software implementations of cryptography, developed by Daniel Bernstein and Tanja Lange, called eBACS (ECRYPT Benchmarking of Cryptographic Systems) [5].

ATHENa allows the characterization of cryptographic algorithms using the FPGAs from multiple vendors, especially Xilinx and Altera. The system also provides different features such as the choice of an FPGA device within a given family, choice of optimum synthesis and implementation options, scanning through multiple placement starting points, and post processing of implementation reports.

The obtained results can be submitted for publication at the project website, together with the exact synthesis and implementation options necessary to reproduce the results. An effort is made to fully characterize the designs using their timing parameters and resource utilization.

Using ATHENa, the designs developed independently by various groups can be compared in a fair, transparent, and uniform fashion, using controlled environment, for multiple families of FPGA devices from various vendors.

1.2 Difficulties and Issues

The difficulties and issues associated with the fair comparison of digital systems designed and modeled using hardware description languages, and implemented using FPGAs, can be divided into two categories: evaluation pitfalls and objective difficulties.

1.2.1 Common Evaluation Pitfalls

Evaluation pitfalls are the mistakes that can be quite easily avoided if the person performing comparison is aware of potential dangers, and exercises appropriate caution and fairness. The evaluation pitfalls are listed below.

- *Taking credit for the improvements in technology:* FPGA manufacturers support multiple classes of FPGA families that evolve over time. One of the main improvements is the process technology used to manufacture FPGAs. The improvement in technology provides a performance gain for implementations on the newer FPGAs over their predecessors.

- *Choosing a convenient but not a fair performance measure:* FPGAs consist of dedicated processing units that could be used to implement logic more efficiently. Using a performance measure such as throughput / CLB slices while implementing logic in dedicated processing units does not constitute a fair comparison.

- *Comparing designs with different functionality:* Similarly comparison of designs with different functionality is not a fair comparison. For example, comparing designs that support both encryption and decryption with a design that performs only encryption.

- *Comparing designs optimized using a different optimization target:* This involves comparing designs that are optimized for different optimization targets, such as speed, area, cost, power, balanced, etc. Each of the criteria might generate results that are substantially different from each other. Such comparisons need to be avoided.

- *Comparing results from the different stages of design flow:* An example of this comparison is comparing the clock frequency after synthesis to the clock frequency after implementation. Synthesis frequency is an estimate of the clock frequency that can be achieved, while implementation frequency is the actual clock frequency that is achieved.

1.2.2 Objective Difficulties

Objective difficulties are more challenging to overcome. The difficulties include

- *Lack of standard interfaces:* The interfaces used by the cryptographic algorithms to communicate with the outside systems might affect the performance.

- *Influence of tools and their options:* The tools used for the FPGA design implementation have a major effect on the generated results. The results depend upon several features and parameters of the tools including tool options. FPGA tools provide sets of options that could be used to control different aspects of the automated synthesis and implementation. The effects of the tool options are design specific. Designs implemented using different sets of options should be compared carefully.

- *Differences in standalone vs. large system performance:* Cryptographic algorithms are usually implemented as a part of a larger design, rather than standalone systems. If a design is implemented as a standalone unit, the tools have the freedom to route through the shortest paths available. However, if a design is implemented as a part of a larger system, the routing resources will be scarce, thus making it difficult to achieve similar performance.

- *Dependence of results on the time spent for optimization:* The time spent on optimization may have a very large impact on the final results. Unfortunately, this time is rarely reported in scientific literature.

- *Design constraints:* The design specific constraints such as timing constraints or area constraints can force the FPGA tools to achieve a better performance. At the same time, the use of constraint files is rarely mentioned in any conference or journal papers.

1.3 Benchmarking Cryptographic Software Vs. Hardware

There exist multiple similarities and differences between benchmarking cryptographic algorithms in software and using FPGAs. The general idea behind ATHENa is acquired from the benchmarking of software algorithms. Some of the properties hold true for both software and hardware environments.

Only a few major vendors are available for both software and hardware platforms. They are Intel and AMD, and Xilinx and Altera respectively. Good quality tools are available for software, while vendors provide most of the tools required for hardware implementation. Complete toolkits or slightly reduced versions are available for free in both cases.

Software can be written to target the specific microprocessor while HDL designs can be targeted to a specific FPGA platform. Similarly, low level optimizations, using assembly language, can be used to achieve better performance in case of software. Hardware designs can make use of low level macros for the same purpose. However, in this case both hardware and software designs might not be portable.

However major differences do exist between the two platforms. Though ATHENa is based on the software environment for benchmarking of cryptographic algorithms many of the ideas cannot be ported in to hardware realm.

The major differences are the performance metrics and optimization targets used for the comparison of the designs. In software, speed is the major parameter. As majority of microprocessors have a fixed clock frequency, software is optimized for minimum number of instructions that result in the fastest execution time. However, in hardware, both speed and area are considered during optimizations and are often traded one for the other. Tools used for the implementation of hardware designs try to optimize the critical path of the circuit resulting in a minimum clock period. In addition, the designs are also optimized for minimum area.

Other differences include the lack of open source designs in hardware. Software implementations of cryptographic libraries are widely available.

1.4 Our Approach

The most important aspects of our methodology are the definition of clear performance metrics and optimization targets for the benchmarking process as well as the collection of specific environment details regarding the design implementation, development of a uniform and practical interface, and generation of results for several FPGA families and converting the results into a single ranking.

Performance of a cryptographic design in hardware is measured in the amount of data processed per a unit of time (throughput) as well as the amount of area occupied and the latency. The throughput is measured in megabits per second. Area is measured in the basic FPGA component units. Latency is measured in seconds. Taking the area into consideration, more meaningful metric is the throughput to area ratio. This metric takes into the account the two main optimization goals of hardware design.

Chapter 2: Motivation and Goals

2.1 Why Cryptographic Algorithms?

One of the main goals of this thesis is to assist with the fair and comprehensive comparison of algorithms that belong to a certain class of digital systems especially where reconfigurable hardware is a feasible and an advantageous means of implementation.

In this thesis, the primary focus is on cryptographic algorithms due to several reasons.

- *Well documented speed ups:* Cryptographic algorithms exhibit well documented speed ups and security gains for FPGA implementations as compared to software implementations.
- *Evolving standards:* Algorithm designers and cryptanalysts are engaged in an everlasting battle to design and crack the cryptographic algorithms. With the algorithms becoming obsolete due to the security issues, designers develop new algorithms to replace the current standards. These algorithms can be implemented and deployed faster when implemented in FPGAs, compared to Application Specific Integrated Circuits (ASICs).
- *Need for fairer evaluation:* With the development of new standards through open competitions, a need for the fair evaluation increases due to the multiple competing algorithms designed by cryptographers around the world.

Starting from the Advanced Encryption Standard (AES) contest organized by NIST in 1997-2000 [1], 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 [6], CRYPTREC in Japan, and eSTREAM in Europe [2].

The evaluation of the candidates in the contests is based on security, performance in software, performance in hardware and flexibility of the algorithm. While security is the most important criterion, it is the most difficult to evaluate and quantify especially due to the short period of time reserved for the contests. Typically after eliminating a fraction of candidates based on security flaws, the rest of the algorithms are evaluated based on their performance in software and hardware.

For example, during the AES contest, held to replace the aging Data Encryption Standard (DES) standard, only five of the original fifteen candidates remained in the competition after their security has been evaluated. Performance in hardware and software has become a decisive criterion of evaluation for the remaining candidates.

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 [7].

At this moment, the focus of the cryptographic community is on the SHA-3 contest, organized by NIST [3], for the development of a new hash function standard. With the contest in its second round, performance in hardware and software become important factors of evaluation. The development of our benchmarking environment is aimed at facilitating the fair and comprehensive comparison among the competing algorithms.

2.2 Goals

As mentioned earlier, the primary goal of this thesis is to facilitate the fair and comprehensive comparison of cryptographic algorithms that are implemented on FPGAs. However, comparison of cryptographic algorithms is not the only important goal of this thesis.

Cryptographic algorithms can be implemented using multiple hardware architectures such as basic iterative, unrolled, pipelined, quasi-pipelined, etc. In addition, different optimization tricks and styles of coding in Hardware Description Languages (HDLs) exist. Different implementation-level optimizations are also 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. Thus, the goal is to provide the fair and comprehensive method for comparison of functionally equivalent architectures and implementations.

Another goal of the project is the identification of an implementation platform which is most suitable for a specific design of a given algorithm. With the presence of different families of FPGAs in the market, it is not an intuitive task to choose a best device, in terms of performance, cost and other factors, for a certain algorithm. A solution is to have comprehensive results available for all the families available in the market.

The results of an FPGA implementation may be a strong function of hardware description languages, tools, and tool versions. Benchmarking such tools and languages is a fourth important goal of the thesis. 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, the goal of the thesis is to facilitate the fair, comprehensive, reliable, and practical comparison of cryptographic algorithms, hardware architectures, FPGA families, as well as the FPGA tools and HDL languages.

Chapter 3: Background

Field Programmable Gate Array (FPGA) is an integrated circuit designed to be configured by the user in the field. It is a combination of programmable elements and interconnects that could be used to realize digital logic.

In this chapter, we look at the details of FPGA architectures from Altera and Xilinx along with the basic design flow that is used in academia and industry.

3.1 FPGA Devices

3.1.1 Major FPGA Vendors and Families

While there are many FPGA manufacturers, only a few manufacturers dominate the FPGA market. Xilinx and Altera take up approximately 90% of the market share, while Actel, Quick Logic, Lattice Semiconductor and other manufacturers occupy the rest. These manufacturers use different technologies to develop the FPGAs. Altera, Lattice Semiconductor, and Xilinx develop FPGAs based on SRAM technology, while QuickLogic and others use anti-fuse technology. Some manufacturers also support multiple technologies. Actel manufactures anti-fuse or flash based FPGAs. However, despite the differences in technology the FPGAs provide similar functionality. While the FPGAs share the basic building blocks their detailed architectures vary depending on the class of applications they are targeting.

manufacturers support multiple classes of FPGA families that are optimized for performance, cost, power consumption, or performance to cost ratio. Families belonging to these classes evolve over time. Along with the process technology used to manufacture the FPGAs, the underlying architecture of the family is also enhanced over time. Families belonging to Altera and Xilinx and the respective process technologies are summarized in the following Tables 3.1 and 3.2.

Table 3.1: Major Xilinx Families

Technology	Low-cost	High-performance
120/150 nm		Virtex 2, 2 Pro
90 nm	Spartan 3	Virtex 4
65 nm		Virtex 5
45 nm	Spartan 6	
40 nm		Virtex 6

Table 3.2: Major Altera Families

Technology	Low-cost	Mid-range	High-performance
130 nm	Cyclone		Stratix
90 nm	Cyclone II		Stratix II
65 nm	Cyclone III	Arria I	Stratix III
60 nm	Cyclone IV		
40 nm		Arria II	Stratix IV
28 nm			Stratix V

In addition, each family consists of several sub-families that target more specific needs of the users. For example, Xilinx low cost FPGA family Spartan 3 has various sub-families that include Spartan-3E, Spartan-3A and Spartan-3A DSP which are logic, I/O, and DSP optimized respectively. The sub-families vary in the amount of resources available on the device.

FPGAs also come in different packages and speed grades. The package defines the physical appearance, size, and the number of pins. The speed grade influences a variety of timing parameters on the FPGA and ultimately defines the performance of the FPGA chip. The speed grade is determined by the fabrication variations that affect the propagation delay of the gates fabricated at any given time.

3.1.2 FPGA Architecture

FPGA architecture varies from vendor to vendor and from family to family. However, the building blocks remain the same. A Look-Up Table (LUT), a basic building block in the FPGAs, can be used to realize combinational logic. A set of LUTs are grouped into logic blocks along with other components like multiplexers and flip-flops on an FPGA. A combination of storage elements and LUTs can be used to realize an almost arbitrary digital circuit.

Along with programmable logic blocks/resources, FPGAs also consist of dedicated structures/blocks that are used to implement logic more efficiently. These structures form logic components that are frequently used in digital systems. These dedicated blocks can be used in place of components implemented through programmable logic blocks to enhance the performance of certain designs. Examples of dedicated blocks are multipliers, adders, accumulators, and other arithmetic and logic components.

Configuration of these different blocks of components varies among vendors. We will look at Altera and Xilinx architecture as they are the major players in the FPGA market.

3.1.3 Xilinx Configurable Logic and Dedicated Resources

Logic

Configurable Logic Blocks (CLBs) are the basic building blocks of Xilinx FPGAs. These blocks can be configured to implement sequential as well as combinational circuits. A CLB is a combination of slices. A slice is a combination of LUTs, multiplexers, registers and fast carry logic. The amount of resources in a CLB varies over different Xilinx families.

In the families prior to Virtex 5, including Spartan 3 and Virtex 4, CLBs consist of four interconnected slices grouped into pairs. Pairs of slices are arranged into two columns, SLICEL and SLICEM respectively. Both of the slices contain two 4-input LUTs, two storage elements, wide-function multiplexers, carry logic, and arithmetic gates. These components are used to provide logic, arithmetic, and ROM functions. In addition, SLICEM also provides functionality to store and shift 16 bits of data [8] [9] [10] [11].

In the newer generation families, including Spartan 6, Virtex 5, Virtex 6, a CLB only consists of two slices. However, each slice consists of four 6-input LUTs. While the newer families have the same number of LUTs the size of the LUT has been increased from 4-input to 6-input. While the architecture remained similar, the sizes of components have been increased.

Table 3.3: Logic Resources in One CLB

Family	# of slices	LUTs	Flip-Flops	Arithmetic and Carry Chains	Distributed RAM	Shift Registers
Spartan 3	4	8	8	2	64 bits	64 bits
Spartan 6	2	8	16	1	256 bits	128 bits
Virtex 4	4	8	8	2	64 bits	64 bits
Virtex 5	2	8	8	2	256 bits	128 bits
Virtex 6	2	8	16	2	256 bits	128 bits

Spartan 6 introduces a SLICEX[8], which inherits the same functionality as SLICEL with the exception that it is missing the arithmetic and carry logic. Table 3.3 presents details of some of the resources available in a CLB.

Dedicated Resources

Along with logic resources Xilinx FPGAs also consist of dedicated resources that are used to enhance performance. Primarily, these structures are faster, smaller and consume less power than if the logic were implemented separately on the FPGA fabric. The dedicated blocks also eliminate the delay that is associated with interconnects, improving overall performance of the designs.

With the exception of certain Spartan 3 families, all the Xilinx families onward are equipped with high performance DSP blocks. These blocks perform various operations including multiplier, multiplier-accumulator (MACC), multiplier followed by an adder, three-input adder, barrel shifter, wide bus multiplexer, magnitude comparator, or wide counter. The DSP blocks can also be cascaded to form a chain without using any CLB resources.

The DSP blocks vary among different families. Spartan 3 DSP block, DSP48A, accepts two 18 bit inputs and produces a 48 bit output. It consists of 18 x 18 bit multiplier, 2 input 48 bit adder among other functions. On the other hand, Virtex 5 DSP block, DSP48E, supports 25 x 18 bit multiplier with 3 input 48-bit adders.

Memory

In Xilinx FPGAs, memory is implemented in two different ways. One method is distributive, while the other is block based.

In a Distributed memory configuration, the individual LUTs are programmed to store data, thus occupying logic resources of an FPGA. This memory configuration is fast, localized, and ideal for small data buffers, FIFOs, or register files. However, this configuration also uses the routing resources that may lead to non-optimal performance of the designs.

FPGAs also provide memory blocks along with other resources. Through the various configuration options, the memory blocks can be configured into RAM, ROM, FIFOs, large look-up tables, data width converters, circular buffers, and shift registers, each supporting various data widths and depths. Similar to the high performance dedicated structures, using the memory blocks instead of distributed memory reduces the congestion on the FPGA ultimately leading to performance improvement.

3.1.4 Altera Configurable Logic and Dedicated Resources

Logic

Logic, in Altera FPGAs, is implemented differently in low cost and high performance FPGAs. The low cost Cyclone families employ a Logic Element (LE) that is built of a 4-input LUT, while the high performance Stratix FPGAs employ an Adaptive Logic Module (ALM) that contains 2 highly configurable Adaptive Look Up Tables (ALUTs).

The primary difference is seen in the configuration of LUTs. The LE consists of one 4-input LUT, along with one programmable register, variety of multiplexers, etc. The LE also provides support to be cascaded in a chain. The LEs can be configured to implement sequential as well as combinatorial circuits. A combination of 16 LEs forms a Logic Array Block (LAB). The architecture is consistent through all the Cyclone families (Cyclone to Cyclone IV).

In case of the high performance FPGAs, logic is implemented through ALMs. An ALM consists of 2 Adaptive LUTs (ALUTs) along with 2 programmable registers, 2 full adders, etc. The ALUTs, with 8 inputs, can be split into combinations of LUT implementations (two 4-input LUTs, a set of 5-input and 3-input LUTs, etc). The ALM architecture is backward compatible for the designs that use 4-input LUTs. With the exception of Stratix II, in all Stratix families (Stratix III through Stratix V), LABs are made-up of 10 ALMs. The Stratix II family LAB consists of 8 ALMs [12] [13] [14] [15].

The area/utilization, in Altera FPGAs, in terms of logic, is measured in either LEs or ALUTs. A conversion factor can be used to convert the amount of LEs to ALUTs.

Dedicated Resources

Similar to the Xilinx FPGAs, Altera FPGAs also consist of dedicated resources that enhance performance of designs. These dedicated resources are implemented in different ways in low cost and high performance FPGAs. Low cost FPGAs consist of multiplier blocks that can be configured into a two 9-bit x 9-bit or one 18-bit x 18-bit configuration. On the other hand, the high performance FPGAs are equipped with DSP blocks.

The DSP block is a combination of multipliers, adders, multiplexers and registers arranged to implement different sizes and combinations of multiplier, multiplier-accumulator functions. The functionality of each new generation of the DSP block is a superset of previous generation.

Memory

Memories can be implemented two different ways. One method is distributive while the other is block based. In a Distributed memory configuration, the storage is achieved using the individual logic blocks. In a block based approach, the memory is packed into blocks.

Altera FPGAs provide different sizes of memory blocks depending on the FPGA family. In the new Stratix V devices the memory block sizes range from 320 bits to 20K bits. These memory blocks can be configured into FIFO buffers, RAM, ROM, etc. Altera memory blocks are designed to interface with the DSP blocks and LABs to provide a high performance.

Memory blocks can be used in place of logic resources to achieve high performance. Also, the use of memory blocks, may lead to better performance by reducing the routing congestion caused by the distributed resources.

3.2 FPGA Design Flow

The design flow outlines different steps involved in the process of generation of FPGA configuration bit-stream from the design specification. The design specification is a collection of information that specifies the requirements of a design. The FPGA configuration bit-stream is used to program an FPGA. Though several FPGA manufacturers exist, the design flow stays consistent across the industry.

3.2.1 General Design Flow

The FPGA design flow is broken down into several steps. These steps include HDL coding, synthesis, implementation and bit-stream generation. Every step of the design flow process can be optimized to a specific target such as minimum power, area, or delay.

Other steps can be included in between these stages to analyze the output from the previous stage of the design flow process. Usually functional verification is performed between each stage to ensure the validity of the output from the previous stage. Additional steps such as static timing analysis and power analysis are performed at different stages of the design flow to provide detailed information about design.

HDL Coding

HDL coding involves the conversion of the design specification into a behavioral HDL description. The two major languages used for circuit description are VHDL and Verilog. The languages provide different capabilities to the user. They could be used to describe the behavior of individual components of the circuits, or could be used to describe a general behavior of a large system through state machines. HDL is a high level language, thus requiring the synthesis and implementation processes to be implemented onto an FPGA.

Synthesis

Synthesis is a process of converting the behavioral HDL descriptions into a netlist. The synthesis process takes as input the HDL design files and a library of primitives that range from simple logic gates to complex FPGA structures. Synthesis process is usually hardware independent and it provides an abstraction layer to the developer to enable the creation of portable HDL sources.

The use of FPGA specific primitives and other information about the size and resources available on FPGA, during synthesis will enable the synthesis process to produce netlist that is optimized for a specific architecture. The output netlist is a complete digital circuit with logic gates and primitives that could be implemented into an FPGA.

Implementation

The next step in the design flow process is the implementation. Implementation process takes the synthesized netlist and generates the programmable netlist that is completely placed and routed. This process is a combination of multiple steps including mapping, placing and routing the design.

- **Mapping**

  The map process breaks the logic functions from the synthesis netlist into smaller blocks such that they fit into the FPGA blocks. The map process fits these sub blocks into targeted FPGA elements such as LUTs, Input Output Blocks and generates a netlist which physically represents the design mapped to the components of FPGA.

- **Placing And Routing (PAR)**

  During the place and route process, the resources assigned during the mapping process are placed into specific locations on the FPGA and the connections between the resources are routed through the routing channels on the FPGA.

  PAR places components into specific locations based on factors such as design constraints (Ex: requested clock frequency), the length of connections, and the available routing resources. The router uses a converging approach to generate a viable solution that routes the design to completion while meeting timing constraints. Both the placer and router algorithms are executed in multiple phases.

The delays associated with interconnects on a large FPGA can be quite significant, so the place and route process has a large impact on the speed of the design. The place and route process attempts to honor timing constraints that have been added to the design. However, if the constraints are too tight, the PAR process will give up and generate a functional implementation that is not capable of operating at the desired speed.

3.2.2 FPGA Tools

Functionally, the FPGA software can be split into two categories: the user interface software responsible for providing an interface for the configuration, starting, stopping, and monitoring the core utilities, and the core utilities, used to perform major computational tasks [1].

Vendors usually distribute software packages that support both functionalities. They are categorized under Electronic Design Automation (EDA) tools. Third-party software is also available to support the individual tasks. For example, Aldec publishes ActiveHDL software that supports multiple vendor platforms, and is used for FPGA design flow configuration and execution. However it does not contain the core technology required to synthesize and implement the designs onto the FPGAs. Majority of the tools are available for free with the slightly reduced functionality compared to the full versions of the tools.

In a way, ATHENa, the software platform being developed as a part of this thesis, can be described as a tool for the configuration and execution of the FPGA design flow using multiple vendor tools. However, it is not the primary goal and much more functionality and features are available as a part of ATHENa.

[1] All the components, in this chapter, are defined in relation to ATHENa point of view. The performance evaluation process involves the study of effects different options have in case of synthesis and implementation. This chapter discusses a small set of the functionalities out of the vast number of features available through the EDA tools.

Each step of the design flow is controlled by a specific tool in the EDA tool package. HDL sources can be synthesized by third-party tools while the implementation specific tasks are performed by vendor specific tools.

All the tools involved in the design flow process, provide a set of options that allow a fine-grain control of the different aspect of the individual process they are involved in. For example, these options could control the effort levels of the different algorithms used to manipulate the circuits. Similarly the options also control more complex operations that affect the results generated by the tools.

3.2.3 Xilinx Design Flow

Xilinx provides a set of Electronic Design Automation (EDA) tools used for implementation of designs on to their FPGAs. The Xilinx ISE toolkit [16] is a set of tools responsible for this process. The ISE itself is the user interface that configures and executes the core utilities. The core utilities include Xilinx Synthesis Technology (XST) that synthesizes the behavioral HDL into a netlist, as well as mapping and place and route tools. In addition, the toolkit also includes numerous other tools for different purposes. Figure 3.1 summarizes the detailed design flow used by the Xilinx FPGA tools.

The list of tools used in the process is specified below. While there are several executables available in the ISE package [17], only the main tools used in the design flow process are covered . (Note: The current description only includes the tools in Microsoft windows operating system.)

- xst.exe - Performs the synthesis of the HDL code and outputs a circuit netlist.
- ngdbuild.exe - Converts the netlist into components according to the format specified in the Xilinx native generic database. The output NGD file is ready to be mapped.
- map.exe - performs the mapping process and outputs a Native Circuit Description (NCD) file.

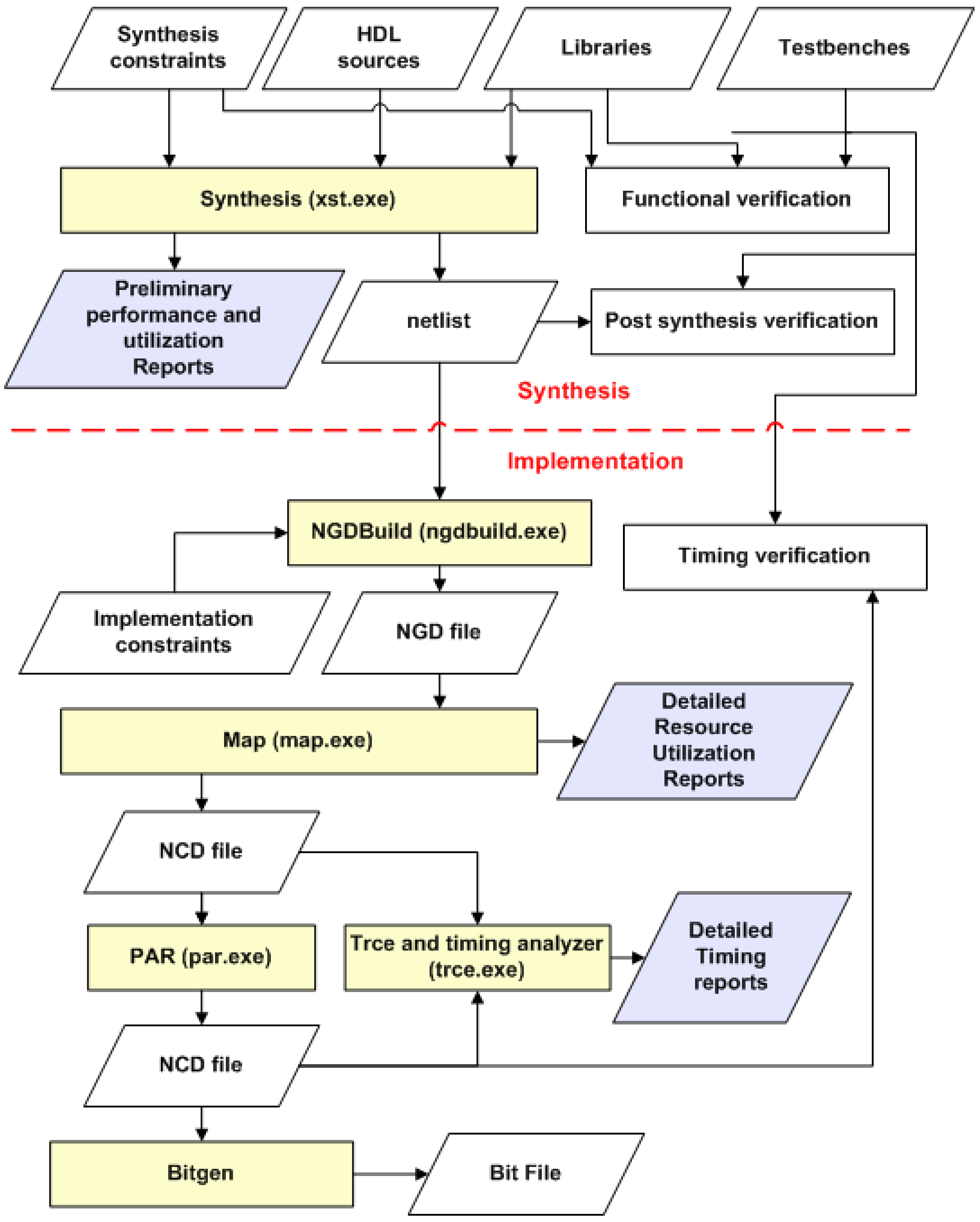


Figure 3.1: Xilinx Design Flow

- par.exe - performs the place and route on the mapped circuit and outputs the NDC file.
- trce.exe - it is a timing analysis tool used on the NCD file to generate detailed reports about the timing.

Execution of Tools

The tools provide several methods of execution control. First method, used by ATHENa, is to configure them through detailed command line options. With the exception of XST all the tools in the Xilinx library accept the options available as command line parameters. XST requires the information to be provided as a separate file. The other method is to provide a project file that contains all the information required for the tool execution. Xilinx ISE GUI creates the necessary project file and passes it on to the tools. The project file container approach is partially used in ATHENa through the use of RUN object (Section 7.3).

Operation in Batch Mode

ATHENa directly interfaces with the tools for their configuration and execution. The generic method of executing each of the tools is given below along with all the parameter description.

- xst.exe [-ifn input file] [-ofn output file]
- ngdbuild.exe [design_name] [ngd_file.ngd] [-option: flag]
- map.exe [input file.ngd] [-o output file.ncd] [-option: flag]
- par.exe [input file.ncd] [output file.ncd] [constraints.pcf] [-option: flag]

Only the required parameters are specified, while other parameters are specified using [-option flag] format. In addition to the report files, the tools output an error number to identify the error information. Detailed information about all the tools and their options is published in the ISE command line guide.

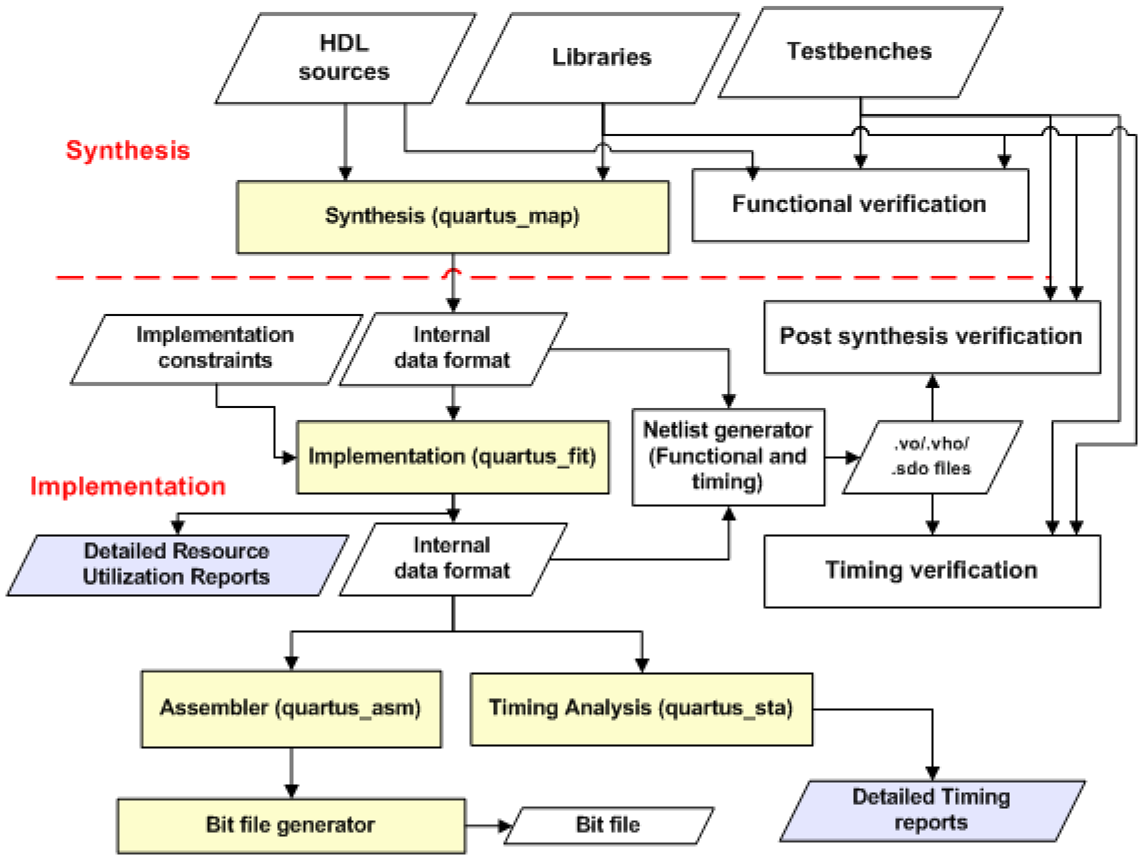


Figure 3.2: Altera Design Flow

3.2.4 Altera Design Flow

Altera EDA tool set designed for the implementation of designs on Altera FPGAs is called Quartus [18]. Quartus also has similar set of capabilities as the Xilinx ISE. It provides a user interface for the configuration and execution of the core tools. Reflecting the general design flow, Altera design flow is broken down into synthesis and implementation. Designs are analyzed and synthesized by quartus_map while the implementation is handled by quartus_fit. Quartus_tan is the timing analyzer, quartus_asm is the assembler and quartus_eda is the netlist writer. Figure 3.2 summarizes Altera design flow.

Circuit data between the different stages is passed using a proprietary file format. The files are converted into simulation compatible files by the quartus netlist writer.

Execution of Tools

Quartus command line executables are designed to be effectively integrated into the batch flow. Quartus tool framework is designed such that the command line executables can be interchangeably executed with the GUI. Thus, the tools use the concept of a container object during the execution. The quartus flow is based on the project file that is used both by the tools and the GUI. The manipulation of the project file is done by the tools and GUI through the different stages of the design flow. However, the quartus tools also allow the specification of command line options. These options are restricted to the major options of the tools.

Operation in Batch Mode

All Quartus tools follow the similar format of command line parameter specification that is shown below.

```
Quartus_tool [–option = flag] [project name]
```

Chapter 4: Previous Work

The main goal of this thesis is to provide support for the performance evaluation of cryptographic algorithms in a fair and comprehensive manner. Different environments exist that address the need for performance evaluation. ECRYPT Benchmarking of Cryptographic Systems (eBACS) [5] is the environment that allows the performance comparison of the cryptographic algorithms implemented in software. Similarly, FPGA vendors have developed tools for the exploration of implementation options. In this chapter, we look at the previous work that has enabled the performance evaluation of cryptographic algorithms.

4.1 eBACS: ECRYPT Benchmarking of Cryptographic Systems

eBACS project was started by Daniel J. Bernstein and Tanja Lange in 2006 [5]. Within the project an open source tool, SUPERCOP, was developed to facilitate comparison of software implementations of cryptographic algorithms.

The general idea is to measure the performance of the software implementations of cryptographic algorithms on different microprocessors. The performance is measured as the number of clock cycles taken to process a unit of data. The basic measurement is the cycles per byte.

As software compilers provide several different options for the code compilation, the option space exploration becomes a problem. The tool supports the choice of best options from among 1200 different combinations of compiler options used to compile the source code of the cryptographic algorithm.

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).

In spite of clear differences among the software and hardware benchmarking, such as performance metrics and optimization targets, the major ideas can be applied in the FPGA domain.

4.2 Option Space Exploration Tools

FPGA vendors have recently started the development of tools for the exploration of implementation options. The major FPGA vendors Xilinx and Altera publish their version of exploration tools named ExploreAhead [19] [20] and Design Space Explorer [21] respectively. ExploreAhead is part of a high level optimization tool called PlanAhead.

Similarly to ATHENa, these software tools allow the exploration of different implementation options based on user defined strategies or predefined strategies that are shipped with the tools. Each strategy corresponds to a set of options for each of the tools involved in the FPGA implementation.

Compared to the exploration tools that target a specific vendor, ATHENa provides the capabilities to explore FPGA devices of multiple 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.

Chapter 5: ATHENa: Overview and Features

To address the issues and difficulties specified in Section 1.2, and to achieve our goals ATHENa (Automated Tool for Hardware EvaluatioN) [4] has been developed. The system facilitates the fair comprehensive, reliable, and automated comparison of algorithms, architectures, hardware platforms, HDL languages and FPGA tools.

5.1 ATHENa Overview

ATHENa provides a platform to conduct comprehensive experiments and compare the results of different cryptographic algorithms through a reliable methodology. The general idea of ATHENa is shown in Figure 5.1 with the data flow direction marked in the order of occurrence.

ATHENa server is the focal point of the environment. The server hosts the ATHENa website and the repository of scripts and configuration files available for download. The server also hosts the database of results submitted by the developers. ATHENa website provides a communication interface between the server and all the other entities in the environment.

As one of the goals of ATHENa is to support the SHA-3 competition [3], ATHENa website provides information about all cryptographic algorithms competing in this contest. This information includes the algorithm specification, reference implementation, and test vectors used for verification of the design. In addition, the website also provides the interfaces and test benches required for the performance evaluation process.

Hardware developers can download these specifications, interfaces and test benches to develop an implementation of a certain algorithm in HDL. The designers can also develop their own interfaces and test benches by themselves following the guidelines in ATHENa

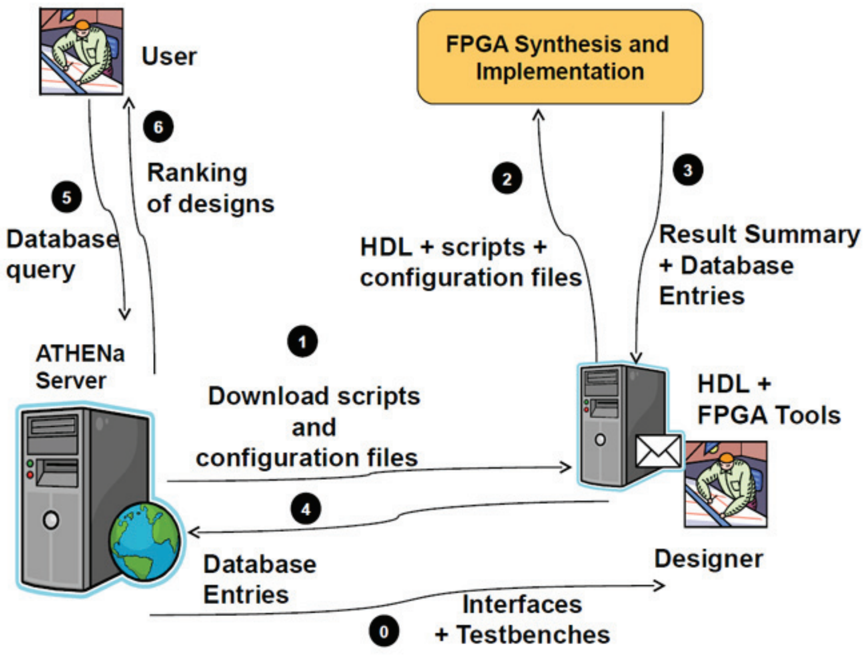

User
FPGA Synthesis and Implementation
6
Ranking of designs
5
Database query
2
HDL + scripts + configuration files
3
Result Summary + Database Entries
ATHENa Server
1
Download scripts and configuration files
HDL + FPGA Tools
4
Database Entries
Designer
Interfaces + Testbenches
0

Figure 5.1: Athena environment overview

developers manual [22]. In that case, the prior download of interfaces is not required. Once the HDL code is ready and its functionally tested through simulation, the performance evaluation process can begin.

Performance evaluation process starts with the developer/user downloading the ATHENa scripts and configuration files from ATHENa website to a local machine. The software required for running ATHENa is a Perl interpreter and the Electronic Design Automation (EDA) tools provided by the FPGA vendors.

The Perl scripts automate the FPGA design flow (Section 3.2). The configuration files contain information about the location of HDL source files, location of tools, target hardware platforms, optimization strategies and other parameters required by the scripts. The user modifies the configuration files such that they contain all the information necessary to configure the scripts. The scripts will execute the EDA tools based on the information provided in the configuration files.

After the execution, a report summary will be generated by ATHENa scripts in the format that is human readable and suitable for database storage. The report contains a summary of results, extracted from the EDA tool reports, in terms of performance, and utilization of the FPGA device. The designer can review the reports and can upload the corresponding database entries to the ATHENa database.

The ATHENa server hosts database of results for various algorithms submitted by multiple developers. These results will be available to the cryptographic community to be reviewed. A reviewer can submit a request to the ATHENa server through the website to acquire ranking or other information related to the designs being reviewed. ATHENa environment does not require the source codes to be published for the reviewers. All the required computations are performed on the user's local machine, thus eliminating the need to reveal the source codes to third parties. The reviewers have the necessary information, such as detailed report of the build environment, to conduct a fair comparison.

5.2 Features

The main features of ATHENa are outlined here.

5.2.1 Multi-vendor and Family Support

ATHENa currently supports Altera and Xilinx FPGAs. Together these vendors account for about 90 percent of the FPGA market. A substantial variability exits between these vendors FPGA architectures. As a result, the ranking of cryptographic algorithms or architectures obtained using devices of one FPGA vendor may not carry over to the devices of another vendor. FPGA vendors also support multiple classes of families that are optimized 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. ATHENa allows us to investigate the dependence of results and rankings of algorithms on these FPGA families.

5.2.2 Automatic Choice of FPGA Device

Performance degradation occurs on FPGAs when the device utilization of a certain design nears 100%. 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. Thus, the ranking of algorithms is dependent on the FPGA device being used.

ATHENa allows a best match option that chooses a specific device for a cryptographic design based on the device utilization factors set by the user. At first, the design is implemented on an arbitrary device of a given family. ATHENa then retrieves the utilization results of the implemented design and calculates the best device based on a device library. This feature also allows the user to specify the utilization factors for not only the logic resources, but as well the dedicated resources such as memory and DSPs.

ATHENa maintains a library with information about all the devices of a given FPGA family and the available resources. These libraries are constantly updated with the emergence of newer versions of vendor tools.

Given the variety of FPGA devices available in the market, ATHENa could determine the best choice of a device for a particular implementation style of an algorithm.

5.2.3 Option Space Exploration

The results generated by the FPGA tools are highly dependent on the choice of options, and design constraints. A large variation in the results can be noticed with different sets of options. ATHENa provides the capabilities to investigate the effects of different options on various families and devices.

Currently ATHENa tackles this task with the use of applications. Applications are extension modules to the Perl scripts that are used to study the effects of different sets of options. A detailed explanation of all the applications is given in the Section 6.2.

5.2.4 Design Verification

Another capability provided in ATHENa is the automated design verification. Currently only functional verification is performed before the execution of FPGA tools. The verification is a necessary step to conclude the correctness of the implemented design. However, the verification step is not required. The verification is implemented in such a way that it is performed automatically for all the designs if the appropriate conditions are set.

Functional verification is based on the testbenches that return a binary answer concluding the passing or failure of a specific design. Sample testbenches are published for the cryptographic algorithms that are competing in the SHA-3 competition.

Designers are responsible for developing testbenches for new algorithms following the generic templates provided through ATHENa website.

5.2.5 Automated Result Generation

ATHENa is capable of executing the FPGA design flow in an automated manner. Due to the comprehensive and time-consuming nature of the executions ATHENa can run for long periods of time without any user supervision.

Chapter 6: ATHENa: User Interface and Core Components

At the core of ATHENa environment is a set of Perl scripts that carry out the performance evaluation process. The configuration of the scripts is done through a text based interface. All information required by the scripts is provided through text files. In addition to the configuration files, the scripts accept as inputs HDL sources, constraint files, and device library files. They produces report files with the results of evaluation. This chapter describes the user interface as well as some of the core components of ATHENa.

6.1 Primary Configuration File

The configuration of ATHENa is achieved through configuration text files. The main configuration file is referred to as design configuration and contains basic information required for the execution.

The design configuration file is split into different sections, which are used to control different features and the process flow of ATHENa. They are global settings, verification settings, and design and implementation settings.

The global settings specify the information about the location of sources, and the workspace directory used to store results. The verification settings specify the information necessary to perform verification of the design. This information includes testvectors, number of clock cycles required to perform the correct verification, etc.

The design and the implementation settings are the crucial part of the configuration. They include settings pertaining to the design such as the top level design entity, clock net name, optimization target, the type of application (Section 6.2), and the different families and devices the design is to be implemented on.

The design configuration file guides the execution of the ATHENa environment.

6.2 Applications

Applications are extension modules to the core Perl scripts that are used to study the effects of different sets of tool options. Given a set of options, an application implements the design using different combinations of these options, while trying to achieve an optimization goal. Applications can be targeted towards achieving different optimization goals.

Each of the applications is executed individually for every FPGA family specified in the design configuration. Additionally, each application is assigned a separate configuration file that specifies all the required information for its successful execution. The details about application configuration files are given in the ATHENa developer guide [22]. ATHENa comes with a set of default applications that are described below.

- **Single run:** A single run is the execution of the FPGA design flow using just one set of options. This is similar to executing the FPGA tools in batch mode, as described in Section 3.2. The options used for the execution of the tools specified by the user in the text based options file are loaded and passed to the tools.

  The single run application acts as a base for all the other applications designed for ATHENa. Applications break down multiple combinations of options into numerous single runs each with a unique set of options.

- **Placement search:** Placement search permits the exploration of the result dependencies on a starting point of placement during the place and route process. The starting point is determined by the place and route tool options. These options are referred to as COST TABLE in Xilinx FPGAs and SEED in Altera FPGAs. The COST TABLE is an integer value between 1 and 100, while SEED is an integer between 1 and 2^{32}. Exploration of the full range of values is computationally prohibitive, especially in case of Altera, so a representative subset of the full range needs to be selected. The COST TABLE and SEED values are specified in the placement search configuration file.

- **Exhaustive search:** Exhaustive search extends the option exploration to include all the possible tool options along with the investigation of multiple target clock frequencies. Since it becomes computationally prohibitive to explore all possible options, exhaustive search allows the exploration of options in two levels to reduce the execution time. The option sets explored at these levels are specified through the exhaustive search configuration file.

Level 1 options are explored while level 2 options remain at their default values. A subset of level 1 options is then chosen according to the optimization criteria, such as speed, area or speed to area ratio. At this point, the level 1 options are kept constant at their best possible values, while the level 2 options are explored.

6.3 Tool Options

Tool option files specify all the information necessary to execute the FPGA tools through ATHENa. The options files follow a hierarchical approach that is focused on the ease of modification by the users. The hierarchy is shown in the sample option description below, thus allowing multiple vendors, tools, options and flags to be described in a single option file.

```
VENDOR_TOOL_OPT
#comment
Option 1 = value 1
Option 2 = value 2
END_OPT
```

ATHENa applications use a unified option file format. Development of new applications only requires the use of abstract functions defined in ATHENa. The data structure of the options files is defined in Section 7.3.

6.4 Library Files

Device libraries have been developed to support different features of ATHENa. The library is a collection of information about the devices of a given FPGA family. ATHENa library files support all major families from Xilinx and Altera and they are organized according to FPGA tool versions currently supported by ATHENa. These libraries are constantly updated with the emergence of newer versions of vendor tools and families. To reduce the size of library files only FPGA devices with the largest packages and highest speed grades are represented in the device library.

For Xilinx FPGAs, the parameters stored in the library files include the number of CLB slices, BRAMs, DSP units, multipliers, and I/O pins per every device. For Altera devices, the library files include the number of Logic Elements or Adaptive Look-Up Tables, total amount of memory bits, DSP units, multipliers and I/O pins.

6.5 Result Files

ATHENa generates a complete report summary at the end of execution. The report is generated both in human readable and machine friendly formats. The human readable files are formatted in ASCII such that the user can easily review them after the execution. The database entries are stored as a comma separated file that can be uploaded to the ATHENa database. Both files share the same information.

The report summary includes the details of families and devices used in the evaluation process, tool options, performance and utilization reports, and an execution time report for each of the runs.

The utilization report provides information about device resources used by the design. These resources are similar to the ones kept in the library; CLB slices, BRAMs, DSPs, multipliers, and I/O pins for Xilinx and Logic Elements, total amount of memory bits, DSPs, multipliers and I/O pins for Altera. The performance/timing report includes information about the requested and achieved clock frequencies per each run on a device. ATHENa also provides the capabilities to calculate throughput and latency from the performance reports by using user provided formulas. These formulas are provided in the design configuration file.

The option report contains the command line options and flags that are used during the execution. Similarly, the synthesis, implementation and total execution time is reported for each run.

In addition, ATHENa also allows the partial results to be viewed during the course of the execution, using a script called report generator. The instruction and details about the report generator are present in the user manual.

6.6 ATHENa Database

ATHENa database provides an interface to store and view the results of all the algorithms submitted by designers.

It stores the information regarding the exact conditions in which the design was evaluated. This information includes the tool options, constraints, tool execution times, etc. The database also stores the performance and utilization results. The lack of information about the conditions of evaluation, while comparing the results, may lead to an unfair comparison. In addition, the database also stores other information about the designs that is not captured by ATHENa. This includes information about algorithm authors/developers, architecture and optimization target, internal details of the architecture, such as the Datapath width, formulas for the throughput and latency calculation, etc.

Chapter 7: ATHENa: Framework and Internal Structure

Chapter 6 gave details of ATHENa user interface and core components. This Section describes the framework that ATHENa is built upon and gives details about how the core components are implemented internally. The process flow of ATHENa is also described in this chapter.

7.1 Choice of the Programming Language

ATHENa software environment requires automation as well as the capabilities for large volume data processing. The automation requirement is necessary due to the need to execute the FPGA design flow without user interaction. In addition, the report files generated by the FPGA tools need to be parsed for the efficient storage and processing of results by the ATHENa database.

Regular expressions are used to identify patterns in the text based report files generated by the FPGA tools. These patterns represent results or other information related to the FPGA design implementation. Along with the regular expressions, other features such as data types provided by Perl make it an ideal language for the ATHENa development.

Other programming languages provide similar functionality, sometimes in more efficient manner. However, in our case, the ease of development is a much more important factor than the computational efficiency of the language. For example, while implementing medium to large designs, the ATHENa executes for a long time. Majority of the time spent in execution is taken by the FPGA tools, while the Perl script that initiated the execution only accounts for a very small fraction of the total execution time. Thus, the computational efficiency of the scripts has not been an important factor in the decision regarding the choice of the programming language.

7.2 ATHENa Framework

ATHENa framework is designed based on two key concepts. They are flexibility and modular design. Flexibility in ATHENa framework will allow the seamless integration of new versions of tools, new vendors, and new families for the currently supported vendors.

To accompany the addition of new tools and vendors, ATHENa follows the concept of a 'run'. A run is an abstract data structure object that facilitates the collection of all the information required to execute the FPGA design flow once. The information includes the tool names, versions, and options, directory structure, optimization target, and other details about the design being implemented.

A run also provides the ability to store the results after the execution. With all the information packed into a single object, a run can be shared through the different stages of ATHENa process flow. The run data structure has also the ability to be expanded with the addition of new information categories, thus minimizing the number of changes performed in the ATHENa core while supporting new vendors and tools. The run data structure is described in detail in Section 7.3.

Another concept the ATHENa framework follows is the modular design principle. The two areas where this principle is visible are the applications and tool option files. ATHENa allows the development of applications by a third party. ATHENa developers guide [22] provides a set of procedures to adhere to, that would allow the third-party applications to be integrated into the system. Designers can use this feature to test their strategies on the designs during the performance evaluation process. Similarly, the tool option files follow a modular format that is shared by all the applications. Parsing of these option files is done by a plug-in module that could be easily replaced.

ATHENa execution framework is based on hierarchical execution. The execution of an application against the run object is called the run. Applications have the ability to generate and execute child runs as required. A run object is associated with each of the child runs and contains the necessary information for their execution. This allows the possibility of building an application stack. In this methodology, applications can be executed in a hierarchical manner rather than as standalone functions. Applications can set the child runs to execute a different application, thus creating an application stack. The run object, with the collection of all the information can be passed down with the collection of best settings from the parent application in the hierarchy. Currently ATHENa only supports standalone application execution, with a one level of hierarchy, where all the child runs are executing single run application.

7.3 Internal Data Structures

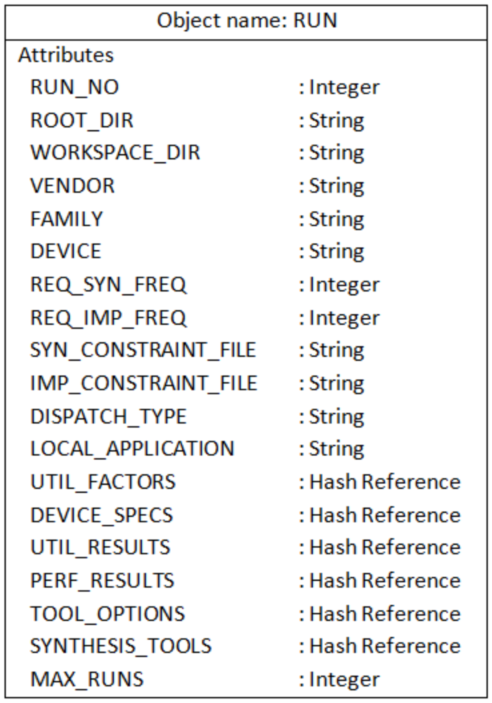
ATHENa consists of several internal data structures that are required for efficient information collection and processing. The main object shared among the different stages of ATHENa process flow is the 'run' object.

The run object is the base on which all the ATHENa applications are built. It contains the detailed information required to execute the FPGA design flow. Figure 7.1 lists all the attributes of the run object.

Majority of the data is represented as strings, integers or other primitive objects. However, some of the information requires the use of a hash data structure. Hashes are associative arrays used to represent data with an associated key. For example, representation of configurable elements in Xilinx device library has two parts associated with it. The CLB slice is the key and the number of slices is the value in the library hash. The advantage of this approach is that, the library hash is flexible to accept different types of configurable elements from multiple vendors; slices for Xilinx or logic elements for Altera.

Information regarding the utilization factors, device information, utilization results and performance results are represented in the run object using hashes. A key difference in the handling of tool options is that they are represented as a hash tree. The run object also provides a set of operators to access and manipulate the specific attributes. The details about these functions are available in the developers' manual.

Tables 7.1 and 7.2 shows the details of the hashes.



Object name: RUN	
Attributes	
RUN_NO	: Integer
ROOT_DIR	: String
WORKSPACE_DIR	: String
VENDOR	: String
FAMILY	: String
DEVICE	: String
REQ_SYN_FREQ	: Integer
REQ_IMP_FREQ	: Integer
SYN_CONSTRAINT_FILE	: String
IMP_CONSTRAINT_FILE	: String
DISPATCH_TYPE	: String
LOCAL_APPLICATION	: String
UTIL_FACTORS	: Hash Reference
DEVICE_SPECS	: Hash Reference
UTIL_RESULTS	: Hash Reference
PERF_RESULTS	: Hash Reference
TOOL_OPTIONS	: Hash Reference
SYNTHESIS_TOOLS	: Hash Reference
MAX_RUNS	: Integer

Figure 7.1: Data structure of the RUN object.

Table 7.1: Hash data structure of Utilization factors, Device specifications, Utilization results is shown here

Object name: **Utilization factors, Device specifications, Utilization results	
KEY : data type	VALUE: data type
*device hardware item 1 : string	# : double
*device hardware item 2 : string	# : double
*device hardware item N-1 : string	# : double
*device hardware item N : string	# : double

* Device hardware item for Xilinx: SLICE, BRAM, DSP, MULT, IO
* Device hardware item for Altera: LE/ALUT, MEMORY, DSP, MULT, IO
** Utilization factors, Device specifications, Utilization results share the same data structure

Table 7.2: Hash data structure of performance results is shown here

Object name: Object name: performance results	
KEY : data type	VALUE: data type
ACHIEVED_FREQ : string	# : double
ACHIEVED_ PERIOD : string	# : double

7.4 Major Program Flow

ATHENa process flow consists of 3 main stages: initial setup, execution, and report generation. The main stages are then subdivided into smaller stages. The following figures illustrate the ATHENa process flow.

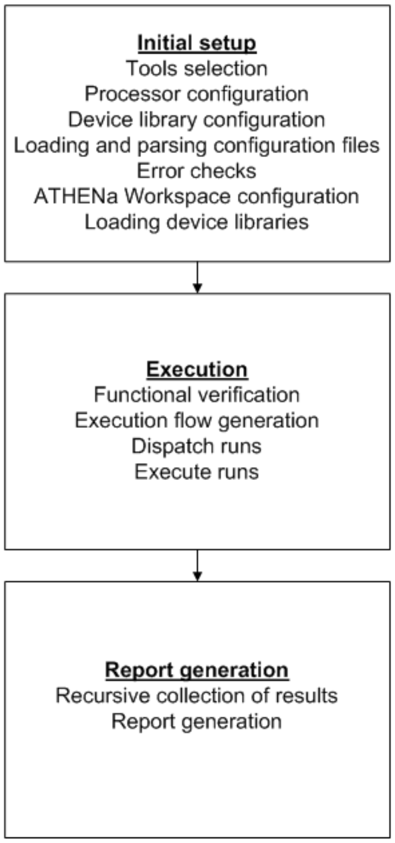


Figure 7.2: Athena process flow breakdown.

The initial setup is required for the configuration and selection of tools, libraries, and workspace as they are the necessary components used for the execution of ATHENa. The configuration and the option files are loaded into the data structures specified in Section 7.3.

ATHENa core execution happens in multiple stages as well. With the required information gathered from the configuration files, ATHENa goes through several steps during execution. First, a functional verification is executed based on the options set in the design

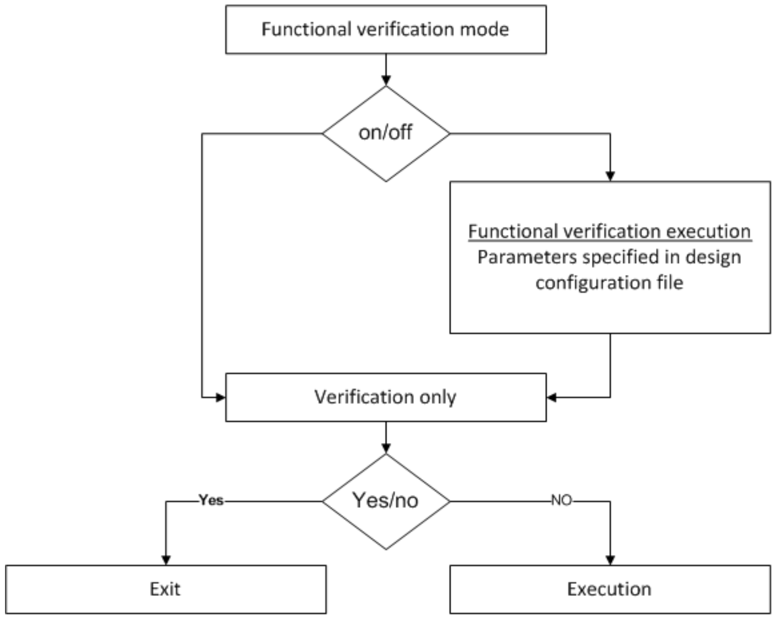

Functional verification mode
on/off
Functional verification execution
Parameters specified in design configuration file
Verification only
Yes/no
Yes
NO
Exit
Execution

Figure 7.3: Functional verification flow

configuration file. The Figure 7.3 shows the control flow of functional verification.

ATHENa then generates the list of runs based on the device and family information provided in the design configuration. The runs are dispatched to the appropriate directories and then executed. The steps in this process are repeated throughout the execution flow. Each of the runs can generate and execute child runs that form a hierarchy model. This process is labeled execution flow generation. Figure 7.4 shows the steps involved in this process.

The execution of individual runs breaks down to several steps. ATHENa supports the choice of best device for a given design. Before ATHENa executes the application, a device check is performed. If the device is not present, ATHENa performs temporary implementation of the design to make a correct choice of a device. A new child run is created and executed with the appropriate device. Similarly applications can create and execute numerous child runs.

The general idea of the execution is presented in Figure 7.5. In this case, the execution stretches four levels. However, an ATHENa process flow can cover anywhere from

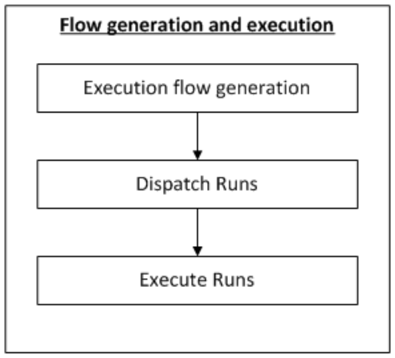


Figure 7.4: Flow generation process of ATHENa

two or many levels. (The maximum number of levels is restricted by the availability of computational resources on which ATHENa is being executed). For example, if the user has specified a single run application with the exact device information, rather than best match, the execution will end at level 2. In the same case, if the user specified the device to be a best match, the process flow would end at level 3. During level 2 ATHENa would find the best choice of the device and generate and execute a new run with the appropriate device.

Detailed explanations of run configuration are given in the ATHENa developer guide [22].

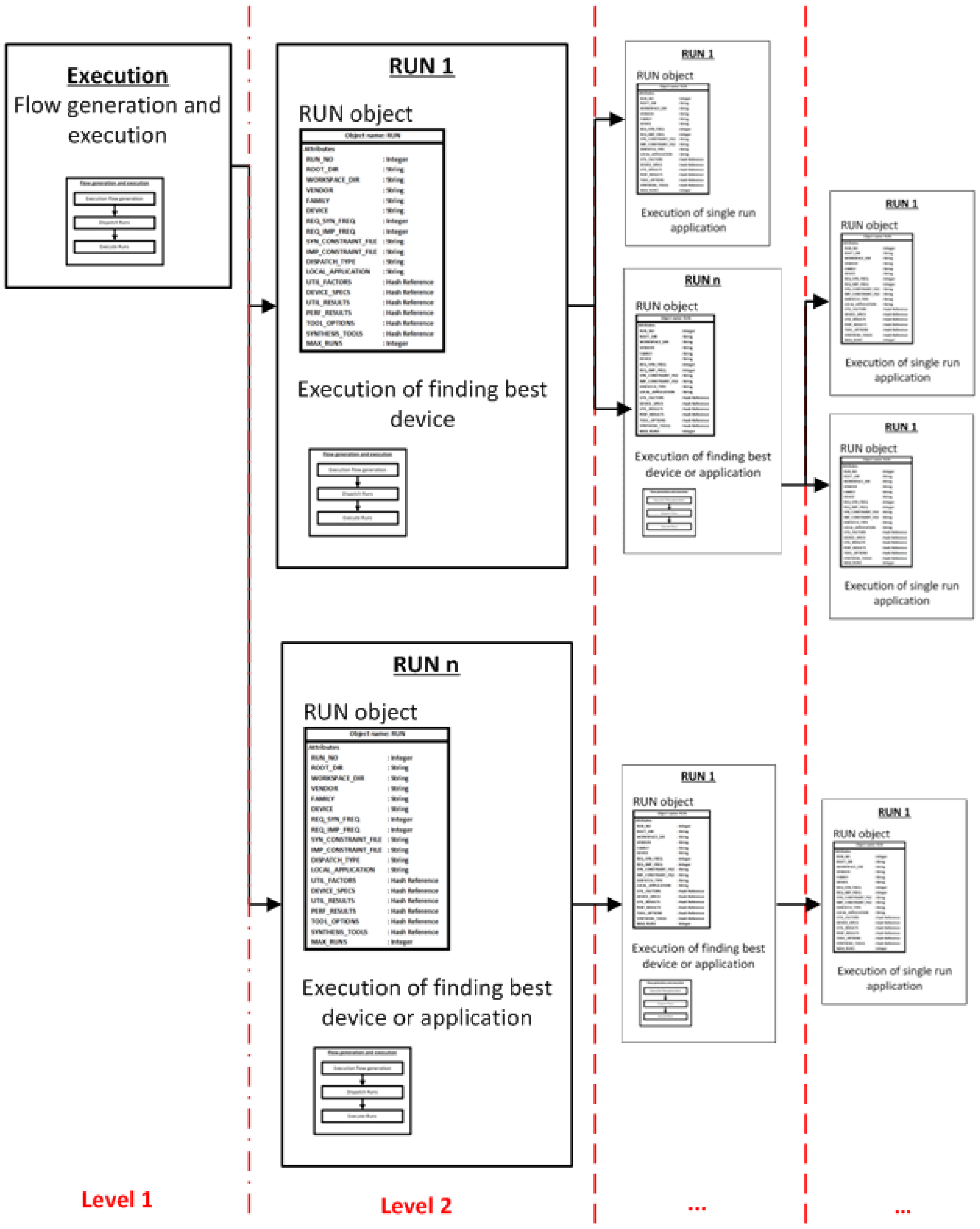

Execution
Flow generation and execution
Flow generation and execution
Execution flow generation
Dispatch Runs
Execute Runs
RUN 1
RUN object
Object name: RUN
Attributes
RUN_NO : Integer
ROOT_DIR : String
WORKSPACE_DIR : String
VENDOR : String
FAMILY : String
DEVICE : String
REQ_SYN_FREQ : Integer
REQ_IMP_FREQ : Integer
SYN_CONSTRAINT_FILE : String
IMP_CONSTRAINT_FILE : String
DISPATCH_TYPE : String
LOCAL_APPLICATION : String
UTIL_FACTORS : Hash Reference
DEVICE_SPECS : Hash Reference
UTIL_RESULTS : Hash Reference
PERF_RESULTS : Hash Reference
TOOL_OPTIONS : Hash Reference
SYNTHESIS_TOOLS : Hash Reference
MAX_RUNS : Integer
Execution of finding best device
RUN n
Execution of finding best device or application
Execution of single run application
Level 1
Level 2
...
...

Figure 7.5: Sample execution of ATHENa environment.

Chapter 8: Case Studies

Two case studies are presented in this chapter that are aimed at demonstrating the capabilities of ATHENa. The first case study demonstrates the techniques and methodologies that could be used to compare cryptographic algorithms using ATHENa. An optimization strategy will be developed to achieve the best throughput to area ratio. The second case study is aimed at the comparison of fourteen round 2 candidates in the SHA3 competition.

8.1 Case Study 1: Demonstration of Meeting Comparison Goals

8.1.1 Designs Evaluated

Two algorithms have been selected to be evaluated in the case study. First algorithm is SHA256, a current cryptographic standard. The second algorithm is Fugue256, currently competing in the contest for the new hash function standard SHA3. Two implementations of SHA256 and one implementation of Fugue256 have been chosen. SHA256 was developed by NSA and was standardized by NIST in 2002. Fugue algorithm specification was developed by IBM in 2008-2009, in response to the NIST call for SHA-3 candidates. Fugue256 can be developed in HDL from the specification.

Out of the several hardware architectures of SHA256 two architectures referred to as the basic loop and architecture with rescheduling are chosen. Basic loop is a straightforward sequential implementation of the algorithm, while the architecture with rescheduling is more optimized. The architecture with rescheduling was developed by Chaves et al. [23] which is optimized for the maximum throughput to area ratio.

Efficient implementations of all three designs have been developed in VHDL by the CERG group at GMU. 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.

8.1.2 Methodologies

The case study is aimed at developing a heuristic optimization strategy that offers an acceptable tradeoff between time spent on optimization and the quality of obtained results. First, a set of experiments will be performed on each of the designs mentioned above. Then an optimization strategy is developed to reduce the time spent on optimization while still achieving acceptable results. First the experiments are conducted on Xilinx FPGAs for the rescheduling architecture of SHA256 algorithm. Then similar experiments are repeated for the other architectures with both Xilinx and Altera FPGAs.

In order to optimize the choice of an FPGA device within a given family, the dependence of maximum clock frequency on the CLB slice utilization needs to be determined. In order to determine the dependence, a parameterized circuit is built that is comprised of a cascade of N SHA256 units, separated by registers. Figure 8.1 shows the idea of the parameterized circuit. A Spartan 3 device for which one unit of SHA256 takes about 3.33% of CLB slices has been chosen. This way, by changing the parameter N, the maximum clock frequency of a circuit could be determined for the CLB utilization that ranges from 3.33% to 96.67%. Figure 8.2 shows the dependency of maximum clock frequency on the CLB slice utilization for the Spartan 3 family. To compensate for the effect of tool options in this experiment all the clock frequencies have been obtained using exhaustive search with 48 sets of options, described below. Based on the dependence shown in Figure 8.2, a threshold of 80% of resource utilization is chosen as a value beyond which the maximum clock frequency deteriorates by a factor larger than 10%.

Using ATHENa in the best_match single_run mode with the MAX_SLICE_UTILIZATION set to 80%, the smallest Spartan 3 device, for which the CLB slice utilization does not exceed 80%, was determined to be xc3s200ft256-5. From this point, all the experiments conducted

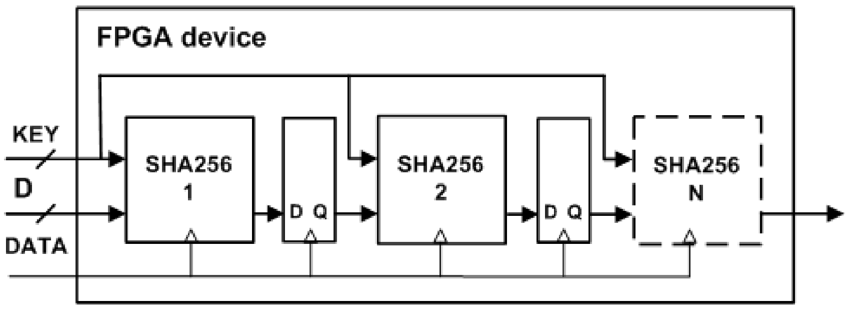


Figure 8.1: A parameterized circuit with N SHA256 units cascaded while separated by registers.

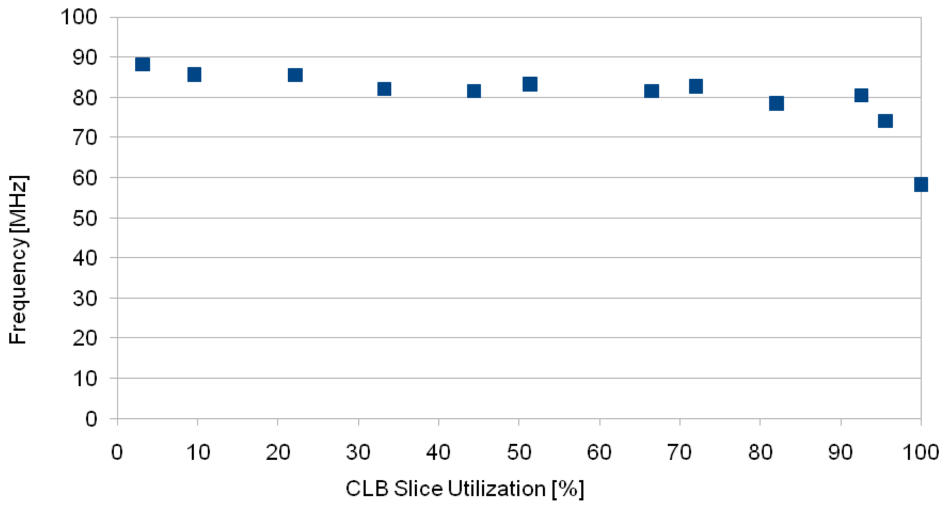


Figure 8.2: Dependence of the maximum clock frequency on the CLB slice utilization for Spartan 3 FPGA.

on SHA256 designs will use this specific FPGA device.

In order to optimize the circuit for the maximum throughput to area ratio, the Exhaustive Search function of ATHENa was employed. Due to the computationally prohibitive nature of exploring all options, a set of options with the major impact on the frequency are chosen. These options are specified below.

- optimization target for synthesis: area, speed
- maximum fan-out: 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

The total number of parameter sets tested was 24 * 3 = 48. The results of this parameter space exploration are shown in Figure 8.3. The data represented in the figure is spread out into three columns. This is due to the synthesis tool options, the first process in design flow, affecting the output significantly. The effect of the rest of tool options is not as significast as the synthesis options. It should also be noted that during this experiment a target clock frequency is not requested. (When the target clock frequency is requested/specified, the PAR tool optimizes the design such that the requested frequency is achieved).

Out of 48 sets of parameters, a set with best ratio of maximum clock frequency to the CLB utilization is chosen. This set corresponds to the options:

- optimization target for synthesis: speed
- maximum fan-out: 100
- optimization target for mapping: area
- optimization effort level for mapping: medium
- optimization effort level for placing and routing: medium

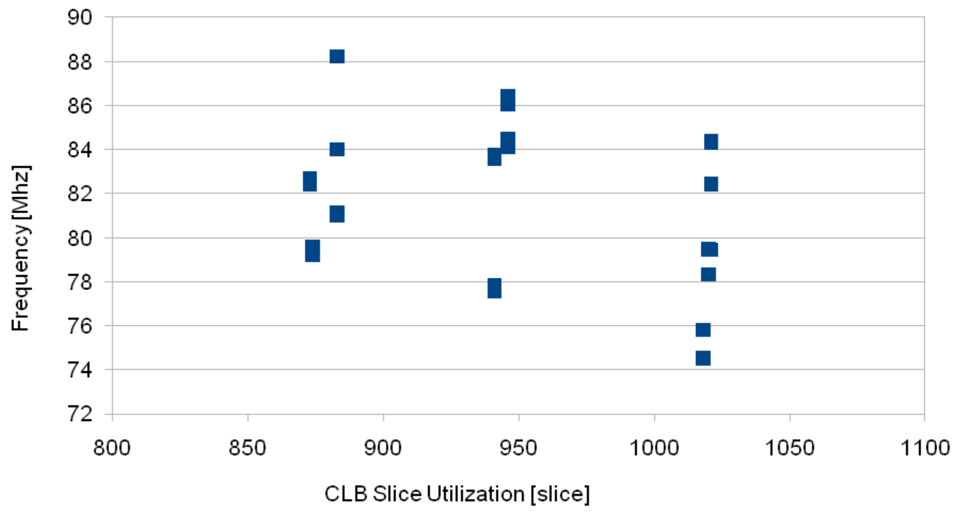


Figure 8.3: Results of the Exhaustive Search for 48 sets of Level 1 parameters.

The total execution time of this phase was equal to about 1.5 hr on the 2.66 GHz Intel Core 2 Duo processor.

The next step is to investigate the effect of 100 different values of Cost Table on the maximum clock frequency. Cost table is the starting point of placer during the place and route process. Depending upon the starting point, the design exhibits performance gain or degradation. Figure 8.4 shows 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 results from default Cost Table value equal to 1. The grey marks on the bars represent the number of results from Cost Table values from the reduced set of Cost Tables 21, 41, 61, and 81. 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.

Another experiment performed is to study the effect of target clock frequency on the obtained clock frequency. Figures 8.5, 8.6, 8.7, and 8.8 demonstrate that the obtained clock frequencies are strong function of target clock frequencies. Target clock frequency represents a frequency that is provided as a requirement for the FPGA tools. In particular,

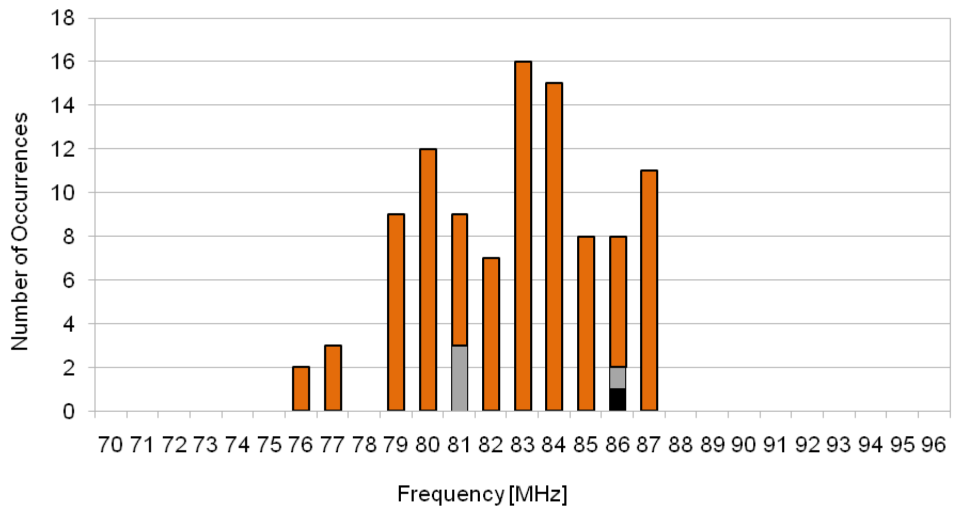


Figure 8.4: Distribution of the actual clock frequencies for the default target clock frequency with 100 values of the Cost Table.

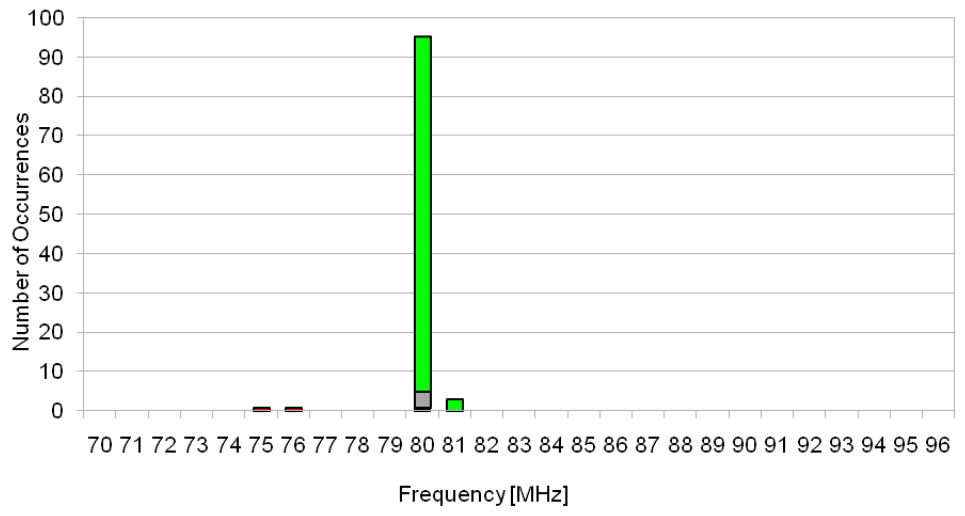


Figure 8.5: Distribution of the actual clock frequencies for the target clock frequency equal to 80 MHz.

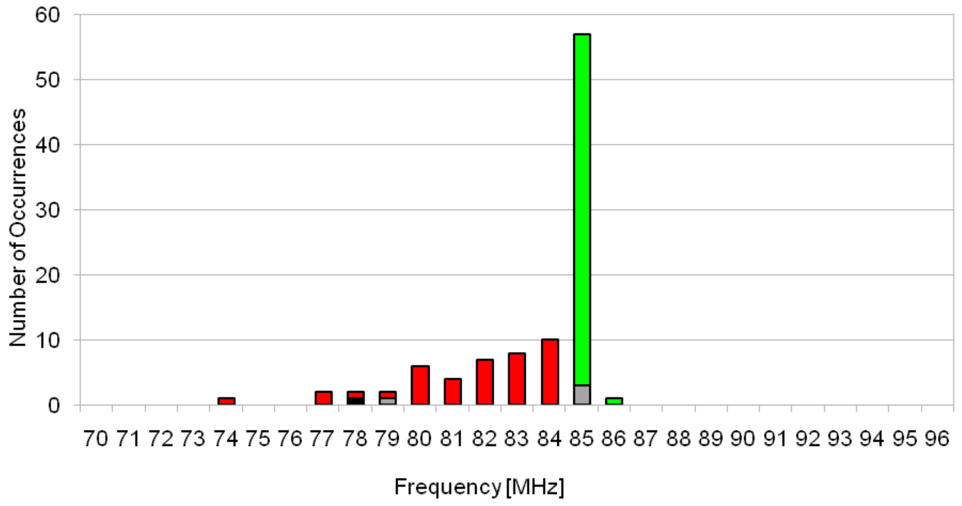


Figure 8.6: Distribution of the actual clock frequencies for the target clock frequency equal to 85 MHz.

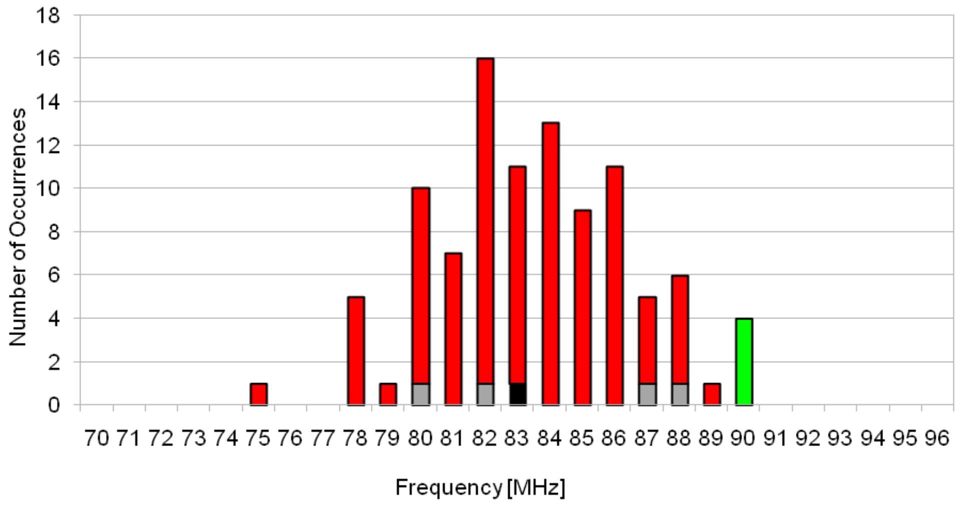


Figure 8.7: Distribution of the actual clock frequencies for the target clock frequency equal to 90 MHz.

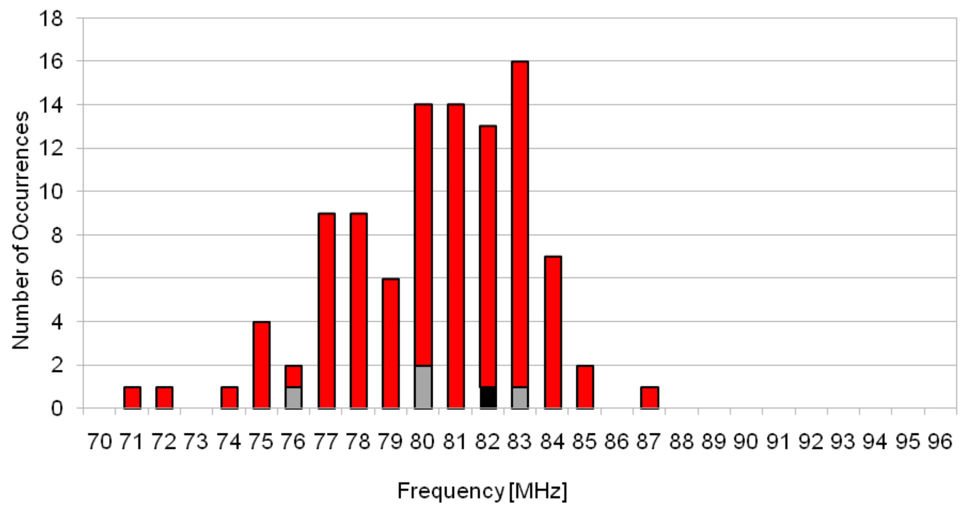


Figure 8.8: Distribution of the actual clock frequencies for the target clock frequency equal to 95 MHz.

when the target clock frequency is either set to default, the spread of the clock frequencies achieved with the different Cost Table values is quite large. A similar behavior appears when the target clock frequency is set higher than the achievable clock frequency. (In this case, the achievable clock frequency is defined such that it is lies with in 20% of the highest clock frequency achieved through the experiments). The results of default clock frequency and high clock frequency settings are demonstrated in Figure 8.4 and 8.8 respectively.

Requesting a target clock frequency that is realistic causes the spread becomes narrower as shown in Figure 8.6. 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. This is demonstrated in Figure 8.5 . In all the aforementioned diagrams, brown bars denote the achieved clock frequencies in case of target clock frequency set to default, green bars denote achieved clock frequencies that are higher than the target clock frequency, and red bars denote frequencies that are lower than the target clock frequency. For example, in Figure 8.6 the target clock frequency is set to 85 MHz. All the clock frequencies under 85MHz are denoted by red, while clock frequencies that are 85MHz and higher are denoted in green.

Table 8.1: Tradeoff between performance improvements and time spent on optimization for the different exhaustive search settings

	Single Run	Full-time exhaustive search	Reduced-time exhaustive search
Frequency MHz	80 MHz	88	90
% improvement over single run	0%	10%	12.50%
Time	Few minutes	2 hours	5 hours

The highest actual clock frequencies were achieved for the case of the target clock frequency equal to 90 MHz, as shown in Figure 8.7. In this case, the maximum clock frequencies found using full-time exhaustive search, reduced-time exhaustive search and single_run are 90 MHz, 88MHz and 83MHz respectively. Thus, the reduced-time exhaustive search gives results falling within approximately 2% from the highest value obtained using full-time search, and it outperforms the single run by 5 MHz (approximately 6%).

Overall, the obtained improvement of maximum clock frequency compared to single run with default values to all parameters, was equal to 12.5% (from 80MHz to 90MHz) for full-time exhaustive search, and 10% (from 80MHz to 88MHz) for reduced-time exhaustive search. The full-time exhaustive search took about 5 hours, while the reduced-time exhaustive search took about 2 hours. However, the improvement is a strong function of FPGA family and the particular circuit. The data is summarized in table 8.1.

In general, the 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.

The next step is to repeat similar experiments on the Altera FPGAs with the Cyclone II family. Similar to Xilinx, the dependency of the maximum clock frequency on the Logic Element utilization needs to be determined.

From the Figure 8.9, a threshold of 80% is chosen as a value beyond which the maximum clock frequency deteriorates by a factor larger than 10%. The best match device chosen by ATHENa is ep2c5f256c6.

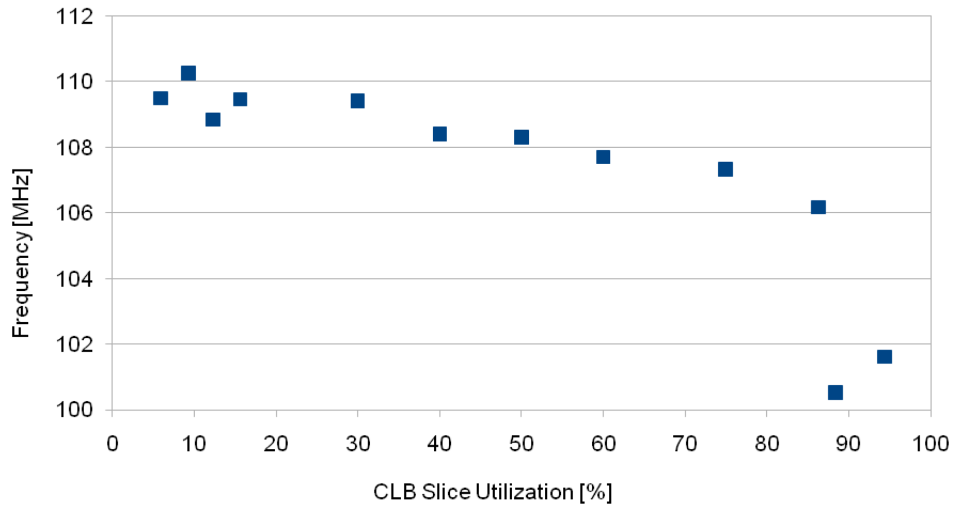


Figure 8.9: Dependence of the maximum clock frequency on the CLB slice utilization for Cyclone II FPGA.

Then the exhaustive search is performed with the option set described below, while setting the target frequency to default.

- optimization target for synthesis: speed, area, balanced
- timing-driven synthesis : yes, no
- optimization effort level for synthesis: std, auto
- optimization effort level for placing and routing: std, auto

The total number of combinations is 24. The results of this parameter space exploration are shown in Figure 8.10. Out of 24 sets of parameters, a set with best ratio of maximum clock frequency to the CLB utilization is chosen. This set corresponds to the options:

- optimization target for synthesis: speed
- timing-driven synthesis : yes
- optimization effort level for synthesis: std
- optimization effort level for placing and routing: auto

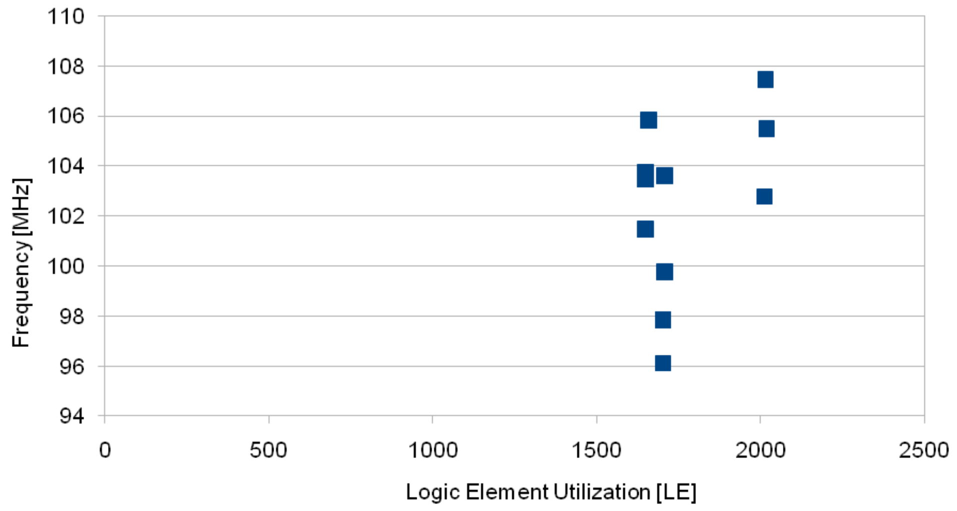


Figure 8.10: Results of the Exhaustive Search for 48 sets of Level 1 parameters.

Next, the analysis on the starting point of placement for the PAR tools is performed. The placement position is referred to as SEED in Altera FPGAs. The values of SEED range from one to 2^{32}-1. A representative subset of 100 values is chosen to perform the analysis. The distributions of the maximum clock frequencies using 100 SEEDs is shown in Figure 8.11. The graphs follow the same format outlined above.

Finally, a target clock frequency analysis is performed on the design. The results of this experiment are shown in Figures 8.12, 8.13, and 8.14. The results from SEED values represented in grey, along with the first results from SEED values shown in black, represent a reduced set of the total 100 values. An interesting observation, unlike in Xilinx FPGAs, is that the maximum achieved clock frequency is a week function of the target clock frequency. All the figures show that the spread of achieved frequencies remains similar regardless of the targeted frequency.

The highest actual clock frequencies were achieved for the case of the target clock frequency equal to 110 MHz, as shown in Figure 8.13. However, the spread of frequencies, in this case, is very similar to the spread where the target clock frequency is set to default. This property remained consistent for all other target clock frequencies. In addition, for all the target clock frequencies reduced SEED set never achieved the clock frequencies that

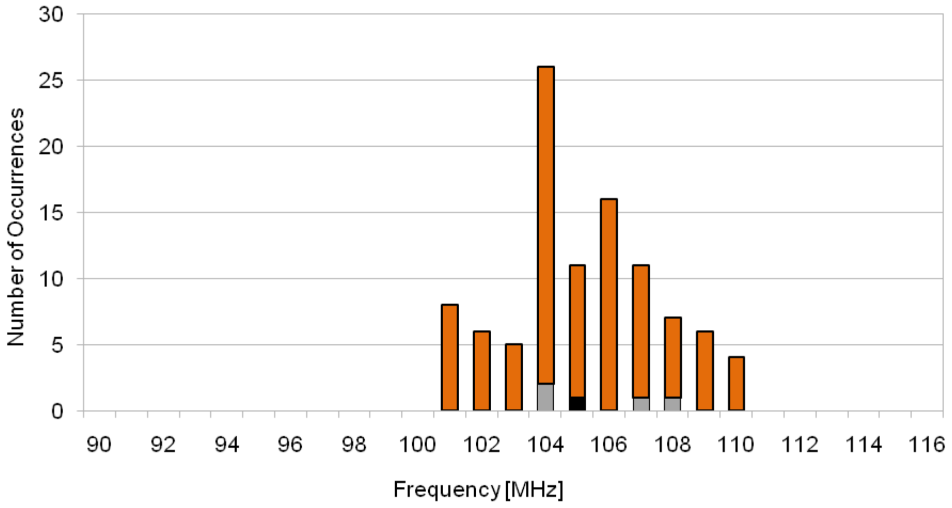


Figure 8.11: Distribution of the actual clock frequencies for the default target clock frequency with 100 SEED values.

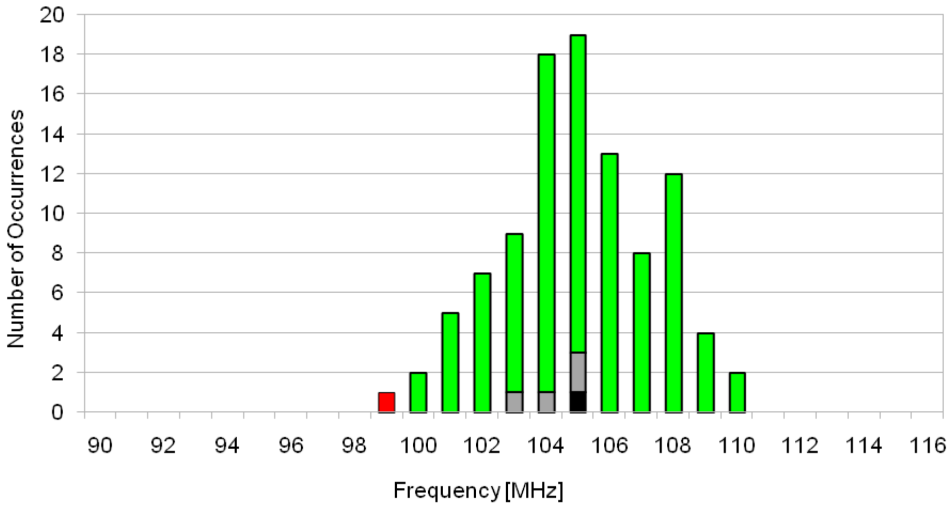


Figure 8.12: Distribution of the actual clock frequencies for the target clock frequency equal to 100 MHz.

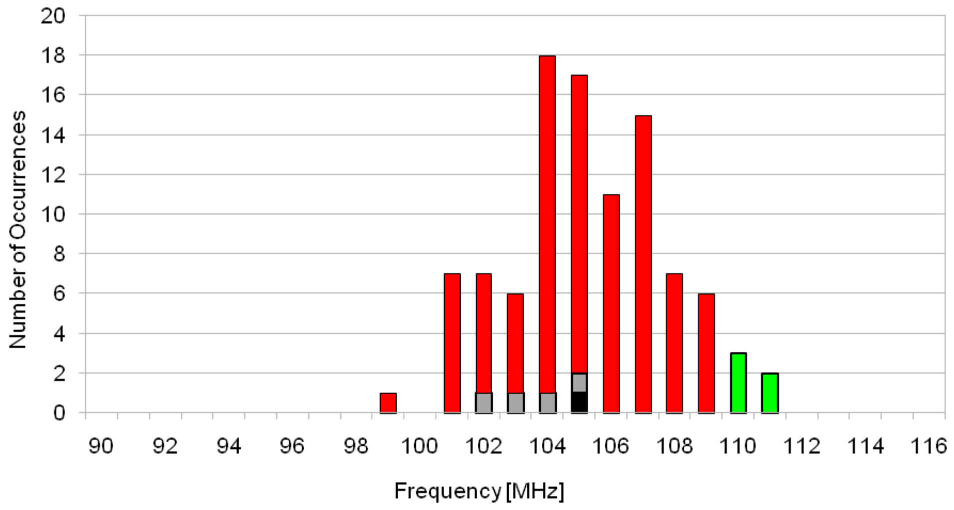


Figure 8.13: Distribution of the actual clock frequencies for the target clock frequency equal to 110 MHz.

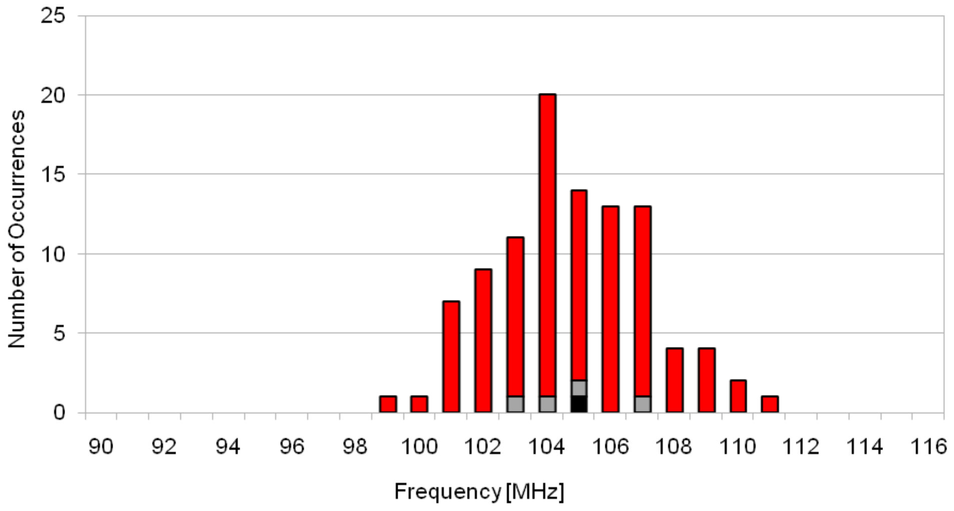


Figure 8.14: Distribution of the actual clock frequencies for the target clock frequency equal to 120 MHz.

were better than the case with the default target frequency. However it is possible that this behavior is very design and tool dependent.

Through the different experiments it is determined that, in case of Altera, the results after place and route were a very weak function of the target implementation frequency. On the other hand, in case of Xilinx, the achieved clock frequencies are a strong function of the target implementation frequency. As a result, different heuristic optimization strategies are created for FPGA devices from Xilinx and Altera.

Xilinx Optimizations

For Xilinx FPGAs, a search for the highest target clock frequency is performed at the beginning. This search involves several single runs of tools, with the target clock frequency first set to the default value, and then gradually increased using a binary search algorithm, based on the corresponding actual clock frequency obtained from a given run. For the highest target clock frequency obtained the exhaustive search is conducted with the number of option sets reduced from 48 to 8 compared to the original experiment. A reduced set of options are chosen such that the different settings of the options have significant effect on the results. The options with lesser effect on the results are eliminated to reduce the execution time. The reduced set of options is specified below.

- optimization target for synthesis: speed, area
- optimization target for mapping: speed, area
- optimization effort level for placing and routing: medium, high

Finally, for the best set of options returned by exhaustive search, placement search is conducted with the number of initial Cost Table values reduced from 100 to 5 compared to the initial experiment. The total number of runs required by this strategy is in the range of 15 to 20. The total number of runs are calculated by adding the number of runs required for the highest achievable frequency search, the number of exhaustive search runs, and the number of Cost Table runs. (Total number of runs = 2 to 7 runs + 8 runs + 5 runs)

Altera Optimizations

For Altera FPGAs, the exhaustive search is applied directly without the search for highest target clock frequency. Similar to Xilinx optimizations, the number of exhaustive search options has been reduced to 12. These options are specified below.

- optimization target for synthesis: speed, area, balanced
- optimization effort level for synthesis: std, auto
- optimization effort level for placing and routing: std, auto

8.1.3 Results and Analysis

The heuristics developed in Section 8.1.2 are applied to the different types of evaluations described in the Section 2.2. The comparisons are between:

- Algorithms: SHA256 vs. Fugue-256
- Architectures: Basic loop vs. Rescheduling
- FPGA platforms: Xilinx Spartan 3 vs. Altera Cyclone II
- FPGA tools: Xilinx ISE v. 9.1 vs. v. 11.1

The results of all the comparisons are summarized in the following table 8.2, 8.3, 8.4, 8.5. In each table, the results are presented in the following order of single run results, optimized results, and the ratio of results respectively. Single run is the basic approach where all the options to the tools are set to default.

All the tables report information regarding frequency, area, throughput, throughput to area ratio, and the execution time. Frequency is expressed in MHz, while area is expressed in CLB slices for Xilinx and Logic Elements for Altera. However, when comparing across platforms, an equivalent measurement for area is required. Thus, the area of Xilinx FPGAs is expressed in terms of logic cells which are approximately half the size of the CLB slices.

Table 8.2: Comparison of two cryptographic hash function algorithms: SHA256 and Fugue-256 using Xilinx Spartan 3

	SHA 256			Fugue 256		
	Single	Optimized	Ratio	Single	Optimized	Ratio
Frequency [MHz]	79.46	88.22	1.11	34.38	40.1	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	1.3	0.28	0.33	1.18
Execution Time [min]	2.15	42.3	18.89	5.16	105.23	20.08

Table 8.3: Comparison of two different hardware architectures of SHA256 using Altera Cyclone II

	Basic Loop			Rescheduling		
	Single	Optimized	Ratio	Single	Optimized	Ratio
Frequency [MHz]	106.47	108.49	1.02	105.5	110.69	1.05
Area [LE]	2291	2216	0.97	2019	2015	1
Throughput [Mbit/s]	838.7	854.6	1.02	831	871.8	1.05
Throughput/Area	0.366	0.386	0.386	0.412	0.433	1.05
Execution Time [min]	0.42	13.02	18.61	0.41	12.58	19.07

The logic cells are approximately equivalent to Logic Elements in Altera. In addition, Throughput is expressed in megabits per second and the ratio of throughput to area is expressed in megabits per second per area unit. Finally, the execution time is expressed in minutes.

The data reveals that Fugue256 outperforms SHA256 in terms of throughput. However, it is inferior in terms of area and the throughput to area ratio. In addition, the optimization of SHA256 improves area and throughput almost equally, while in Fugue, it affects practically only throughput. Though the execution time is relative large for Fugue256, compared to SHA256, the optimization ratio of the execution time is comparable for both designs.

The results of comparison between basic loop architecture and the architecture with rescheduling are shows in table 8.3. These architectures are implemented on Altera Cyclone II device.

In terms of frequency, the basic loop performs better than the architecture with rescheduling in single run. It is surprising that the architecture that is not optimized outperforms the optimized architecture. However, the rescheduling architecture has better performance after

Table 8.4: Comparison of two different target hardware platforms: Xilinx Spartan 3 and Altera Cyclone II for SHA256 (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	Optimized	Ratio	Single	Optimized	Ratio
Frequency [MHz]	79.46	88.22	1.11	105.5	110.64	1.05
Area [LC or LE]	2040	1776	0.87	2019	2015	1
Throughput [Mbit/s]	625.9	694.9	1.11	831	871.8	1.05
Throughput/Area	0.312	0.391	1.28	0.412	0.433	1.05
Execution Time [min]	2.15	42.3	18.89	0.51	14.2	17.27

Table 8.5: Comparison of two different versions of tools: Xilinx ISE Design Suite v.11.1 vs. v. 9.1 for SHA256 (architecture with rescheduling)

	Xilinx ISE v. 9.1			Xilinx ISE v. 11.1		
	Single	Optimized	Ratio	Single	Optimized	Ratio
Frequency [MHz]	77.87	92.58	1.19	79.46	88.22	1.11
Area [CLB slices]	1020	873	0.87	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
Execution Time [min]	2.17	42.2	18.24	2.15	42.3	18.89

the optimizations. This demonstrates that the application of option exploration on designs could lead to better performance. The area of both architectures are quite comparable.

Table 8.4 summarizes the data of the comparison of SHA256 algorithm on two different FPGA architectures, Xilinx Spartan 3 and Altera Cyclone II. The two architectures belong to the same generation and class. Both families are optimized for low cost and are manufactured using 90 nm technology. In this comparison, Cyclone II outperforms Spartan 3 in terms of frequency, throughput and throughput to area ratio. The difference between the families decreases with the optimization. The ratio of improvement in throughput for Spartan 3 is higher than of Cyclone II. However, this it is possible that the results are design dependent. Therefore the conclusions might not hold true for other architectures of the SHA256.

The last comparison is between two different versions of Xilinx ISE tool, Xilinx ISE 9.1 and Xilinx ISE 11.1. The comparison is made by implementing the SHA256 architecture with rescheduling. For a single run, the newer version of tool performed better. However, with the optimizations, the older versions of ISE outperformed the new version. This is an unexpected behavior since never version of tools should be optimized to perform better than their predecessors. However, it is possible that the tools are optimized to target newer generations of FPGA families and the optimizations may not carry over to the previous generations of FPGAs. The area and the execution time remain consistent with the different versions.

In summary, 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.

8.2 Case Study 2: Comparison of 14 Round 2 SHA-3 Candidates

Case study 2 presents the performance evaluation of 256 architectures of the 14 round 2 candidates participating in the SHA3 competition along with the current standard, the SHA2 algorithm. The case study is aimed at improving the results of the implemented designs using just the optimization techniques available in ATHENa. All the designs are implemented and optimized on Xilinx Virtex 5 FPGAs using ATHENa.

Initially, the designs are implemented using the default options (single run application in ATHENa). Subsequently, the designs are implemented using the different optimization techniques available in ATHENa (exhaustive search, placement search, frequency search). The algorithms are evaluated in terms of their area, throughput, and throughput to area ratio. The relative improvement in the results by using ATHENa optimizations over the default options is shown in the figures 8.16 and 8.15.

The algorithms are listed on X-axis, while their relative improvement is shown on Y-axis. Area is represented in blue, throughput in orange, while throughput to area ratio is represented in yellow.

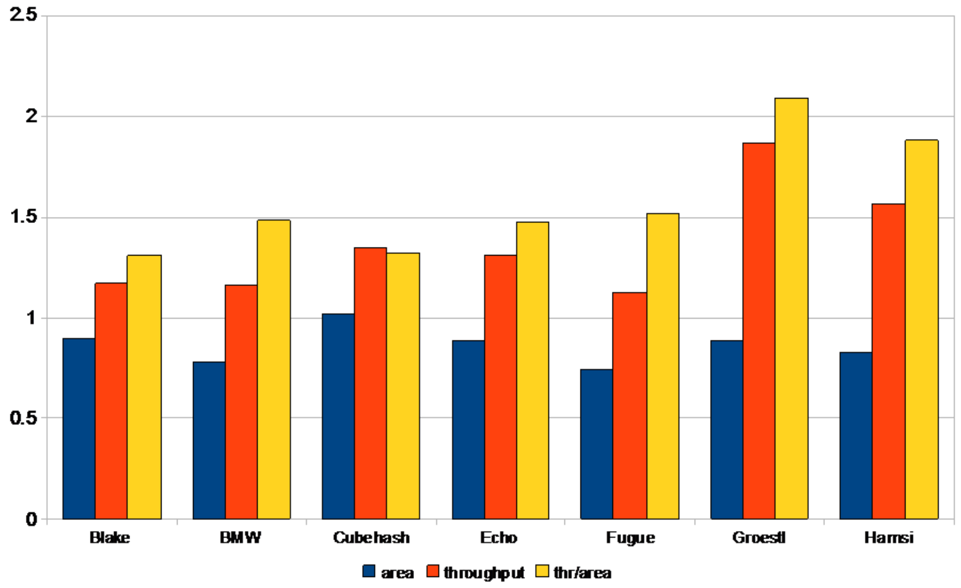


Figure 8.15: Relative improvement of throughput to area ratio of SHA3 candidates using ATHENa optimizations over the implementations without optimizations

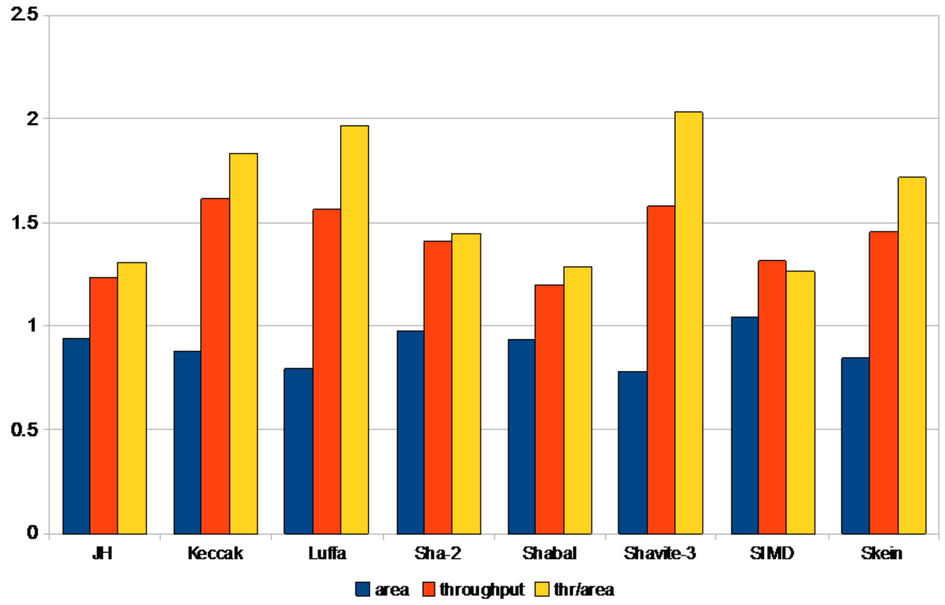


Figure 8.16: Relative improvement of throughput to area ratio of SHA3 candidates using ATHENa optimizations over the implementations without optimizations

Table 8.6: Relative improvement of throughput to area ratio of SHA3 candidates using ATHENa optimizations over the implementations without optimizations

Relative improvement range of throughput to area ratio	Algorithm
1.8 - 2.1	Groestl, Hamsi, Keccak, Luffa, Shavite-3
1.6 - 1.8	Skein
1.4 - 1.6	BMW, ECHO, Fugue, SHA-2
1.25 - 1.4	Blake, CubeHash, JH, Shabal, SIMD

The figures show a significant improvement of results for certain algorithms. Luffa, Shavite-3, Skein, BMW, Fugue and Hamsi achieved a considerable area improvement while Keccak, Luffa, Shavite-3, Groestl, and Hamsi achieved considerable improvement in throughput.

However, in some cases, a decline in the quality of results is also seen. CubeHash and SIMD implementations resulted in the area increase without significant improvement in throughput or throughput to area ratio. The designs can be categorized based on their improvement in terms of throughput to area ratio. The different categories are shown in the table 8.6.

Out of the fifteen algorithms evaluated, five algorithms showed a significant improvement in the throughput to area ratio. Similarly, the algorithms toward the lower end of the spectrum also achieved a performance gain over the FPGA implementations with default options. However, it should be noted that these improvements are specific to the devices of a given family and may not carry over to the devices of other families.

The improvement in results shows that the exploration of tool options is an important aspect of performance evaluation process. ATHENa presents a viable solution for the performance improvement in the evaluation process.

Chapter 9: Conclusions and Future Work

ATHENa, open-source environment, is presented as a solution that overcomes the difficulties involved with, and facilitates a fair, comprehensive, reliable, and practical benchmarking of digital systems using FPGAs from various vendors. The case studies demonstrate the viability of ATHENa as a tool for performance evaluation in a situation where a fair comparison of algorithms, implemented on FPGAs, is necessary.

However, the environment is still in the early stages of development. Addition of new features will make ATHENa a more reliable and user friendly environment that contributes to the cryptographic community. The major new features would include the support for:

- *Additional FPGA vendors:* Designers may be discouraged, from the use of ATHENa environment, if their target FPGA vendor is not available for the comprehensive evaluation. To avoid this, additional vendors like Actel and Lattice Semiconductor need to be supported.

- *Additional EDA tools:* Third-party FPGA tools are available that target the FPGAs of different vendors. The comprehensive analysis of algorithms would require the addition of tools other than those provided by the FPGA vendors.

- *Heuristic algorithms:* With the large sets of options available for each of the FPGA tools, the exploration of all the combinations of options becomes computationally prohibitive. Thus, a set of heuristic algorithms targeted at minimizing the execution time by eliminating sets of options could help the performance evaluation process.

- *Different operating systems:* 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.

- *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. A GUI to assist with preparation of configuration files would be beneficial.

On the other hand, major features already available in the ATHENa environment will assist with the comparison of cryptographic algorithms. These features are:

- *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:* ATHENa does not require the designers to reveal 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.
- *Single point of contact:* ATHENa 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.

At this point, the biggest test for ATHENa is the evaluation of all the candidates submitted to the SHA3 contest organized by NIST. After the contest, ATHENa will still serve the cryptographic community by providing the comprehensive results for cryptographic algorithms. Researchers will benefit from the capabilities of ATHENa, to compare algorithms in a fair, comprehensive and reliable manner. Similarly, Designers will benefit from the capability of comparing results of algorithms on variety of FPGA families 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 the FPGA 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 ATHENa. In addition, ATHENa could also be ported to support algorithms of different classes of digital systems such as DSP or digital communications.

Bibliography

[1] J. Nechvatal, E. Barker, L. Bassham, and William, “Report on the development of the advanced encryption standard (aes).” FIPS Publication 197: National Institute of Standards and Technology (NIST), Oct 2000, http://csrc.nist.gov/archive/aes/round2/r2report.pdf.

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

[3] *Cryptographic hash algorithm competition.* [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/

[4] *ATHENa Project Website.* [Online]. Available: http://cryptography.gmu.edu/athena/.

[5] *eBACS: ECRYPT Benchmarking of Cryptographic Systems.* [Online]. Available: http://bench.cr.yp.to/

[6] B. Preneel, A. Biryukov, and C. D. Canniere, *Final report of European project number IST-199-12324 named New European schemes for signatures,integrity, and encryption-NESSIE*, New European schemes for signatures,integrity, and encryptionNESSIE, April 2004, https://www.cosic.esat.kuleuven.be/nessie/Bookv015.pdf.

[7] 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.

[8] X. Inc., *Spartan-6 FPGA Configurable Logic Block*, 1st ed., Feb 2010. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug384.pdf

[9] ——, *Spartan-3 Generation FPGA User Guide*, 1st ed., December 2009. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug331.pdf

[10] Xilinx, *Virtex-4 FPGA User Guide*, 2nd ed., December 2008. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug331.pdf

[11] ——, *Virtex-5 FPGA User Guide*, 5th ed., May 2010. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug190.pdf

[12] A. Inc., *CycloneII Architecture*, 3rd ed., Feb 2007. [Online]. Available: http://www.altera.com/literature/hb/cyc2/cyc2_cii51002.pdf

[13] Altera, *Cyclone IV Device Handbook*, 1st ed., July 2010. [Online]. Available: http://www.altera.com/literature/hb/cyclone-iv/cyiv-5v1.pdf

[14] A. Inc., *Stratix III Device Handbook*, 2nd ed., July 2010. [Online]. Available: http://www.altera.com/literature/hb/stx3/stx3_siii5v1.pdf

[15] ——, *Stratix V Device Handbook*, 1st ed., July 2010. [Online]. Available: http://www.altera.com/literature/hb/stratix-v/stx5_5v1.pdf

[16] *ISE Webpack Design Software.*

[17] X. Inc., *Xilinx Command Line Tools User Guide*, 11st ed., December 2009, application Note. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/devref.pdf

[18] *Quartus II Subscription Edition Software.*

[19] M. Goosman, R. Shortt, D. Knol, and B. Jackson, "Exploreahead extends the planahead performance advantage," *Xcell Journal*, pp. 62–64, 2006.

[20] R. Shortt, D. Knol, and B. Jackson, "Exploreahead: A methodical approach to improved qor through implementation tools," Jan 2006. [Online]. Available: http://www.xilinx.com/bvdocs/wp_exploreahead.pdf

[21] A. Inc., *Quartus II Handbook Version 10.0*, 10th ed., July 2010.

[22] V. Amirineni, *ATHENa Developers Guide*, 1st ed., July 2010. [Online]. Available: http://cryptography.gmu.edu/athena/index.php?id=download

[23] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis, "Cost-efficient sha hardware accelerators," in *Very Large Scale Integration (VLSI) Systems*, Aug 2008, pp. 999–1008.

Curriculum Vitae

Venkata Amirineni was born on May 5th, 1987 in Guntur, India. He received his Bachelor of Science degree in Computer Engineering from George Mason University in May 2008 and currently pursuing his Masters of Science degree in Computer Engineering at the same university. He is a Research student and part of Cryptographic Engineering Research Group (CERG). His research interests include system integration, performance analysis, and FPGA design.