Hardware Evaluation of SHA-3 Candidates

Sinan Huang

Thesis submitted to the faculty of the Virginia Tech Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science

In

Computer Engineering

Leyla Nazhandali, Co-Chair

Patrick Schaumont, Co-Chair

Sandeep Shukla

May 4, 2011

Blacksburg, Virginia

Keywords: Hardware Evaluation, SHA-3, Security, Cryptography

Hardware Evaluation of SHA-3 Candidates

Sinan Huang

(Abstract)

Cryptographic hash functions are used extensively in information security, most notably in digital authentication and data integrity verification. Their performance is an important factor of the overall performance of a secure system. In 2005, some groups of cryptanalysts were making increasingly successful attacks and exploits on the cryptographic hash function, SHA-1, the most widely used hash function of the secure hashing algorithm family. Although these attacks do not work on SHA-2, the next in the series of the secure hashing algorithm family, the National Institute of Standards and Technology still believes that it is necessary to hold a competition to select a new algorithm to be added to the current secure hashing algorithm family. The new algorithm will be chosen through a public competition. The entries will be evaluated with different kinds of criteria, such as security, performance and implementation characteristics. These criteria will not only cover the domain of software, but the domain of hardware as well. This is the motivation of this thesis.

This thesis will describe the experiments and measurements done to evaluate the SHA-3 cryptographic hash function candidates' performance on both ASIC and FPGA devices. The methodology, metrics, implementation details, and the framework of the experiments will be described. The results on both hardware devices will be shown and possible future directions will be discussed.

Acknowledgements

I would like to thank Dr. Leyla Nazhandali for her guidance and support throughout my research process. I would like to also thank Dr.Schaumont for his guidance regarding my thesis.

I would also like to thank Dr. Sandeep Shukla for their participation in my thesis committee.

I would like to thank my friends in CESCA – Michael Henry and Steve Griffin as they prepared the logistics that made my research possible, Dinesh Ganta, Meeta Srivastav, Kanu Priya and Lalleh Rafeei for their friendly guidance and assistance. Finally, I would like to thank Eric Guo for his tireless and patient guidance during our collaboration.

Last but not the least, I would like to thank my parents Dr. Jianhua Huang, Mrs. Xuefen Wang and my family for their continued encouragement and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past decades, the semiconductor industry has grown rapidly. As engineers and scientists were able to place more and more transistors onto the die of the chip, computers have become more powerful and ubiquitous. As computers increasingly become a part of people's lives, trustworthy security systems must be in place to protect sensitive information from being stolen and used against its original owners' will.

In 2005, several groups of cryptanalysts were able to continuously weaken the widely used SHA-1, which is implemented in TLS and SSL, PGP, SSH, S/MIME and IPsec [1] [2] [3] [4] [5] [6]. However for now, these attacks do not extend to SHA-2, the next in the series of the family of SHA. Nonetheless, the National Institute of Standards and Technologies still believes it is prudent in the long run to develop one or more hash algorithms through a public competition. The winner of this competition will be titled "SHA-3". Since the hash algorithms are a small part of a larger system, their implementation performance must be evaluated [7].

In March 2008, Intel proposed to make an extension to the x86 instruction set architecture to include the Advance Encryption Standard Instruction Set to improve the speed of applications performing encryption and decryption with AES instructions [8]. This shows that to improve performance, other cryptographic algorithms may be implemented in hardware as well. Therefore, hardware performance analysis of those hash candidates is necessary for a complete evaluation.

1.1 Contribution

In the first part of this thesis, the methodology, tool flow, metrics, experimental platform, and results regarding the hardware evaluation of the cryptographic algorithms will be described.

In the second part of this thesis, additional experiments will be presented, including the optimization of a cryptographic algorithm and a study of the effects of technology scaling.

1.2 Thesis Organization

Chapter 2 gives some background information regarding cryptographic hash and the motivation behind this project. It provides some basic information with regard to ASIC, FPGA and each of the SHA-3 candidates.

Chapter 3 discusses the process of the experiments for the evaluation of hash candidates on ASIC as well as on FPGA.

Chapter 4 shows the results of the experiments.

Chapter 5 explores some additional experiments for this project, such as optimizations of algorithms and studying the effects of technology scaling.

Finally chapter 6 gives the conclusion and discusses possible future works to be done on this project.

Chapter 2

Background Information

In this chapter, some basic background information regarding to cryptographic hash functions, such as its description, property and applications will be discussed. The characteristics of the hashing functions will be explained through examples. A description of SHA as well as the motivation behind the competition of SHA-3 will be given. After that some commonly used modules and their implementation characteristics will be shown. In the end, an overview of the SHA-3 candidates and the hardware devices they were implemented on will be given.

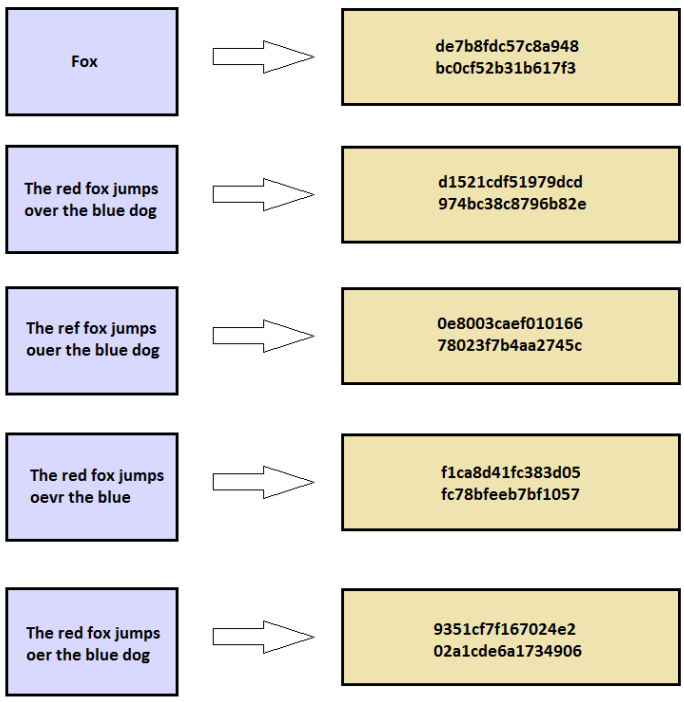

Figure 2.1: A cryptographic hash function (in this case MD5) at work. Note that a slight change in the message should have drastic effects on the output.

2.1 What are Cryptographic Hash Functions?

2.1.1 Description

A cryptographic hashing function is a deterministic procedure that receives a message of arbitrary size and returns a fixed size block of data often called a digest. Figure 2.1 presents an example of a cryptographic hash function, the MD5 hash. Depending on the source of the information, the definition or requirement for a cryptographic hashing function may be different to a certain degree; however, the most common characteristics or desired properties for a secure hashing function *H* with input message *x* are as follows:

- *H(x)* is relatively easy for any given message *x*, so that the implementations of *H* in both hardware and software can be efficient.
- For any given hash digest *d*, it is computationally impractical to find a message *x* such that the hash digest of *x* is the same as the hash digest *d*. This is to ensure that the hash function is one-way.
- For any given message *x*, it is computationally infeasible to find another message *y*, such that the message *y* is not the same as message *x* but the hash digest of *y* and *x* are the same. Some sources referred to this as weak collision resistance.
- It is computationally infeasible to find two different messages, *x* and *y* such that their hash digests are the same. Some sources referred to this as strong collision resistance.

These are common properties, or ideal characteristics, of a hash function. These properties can infer a lot of other characteristics about cryptographic hash functions but it is beyond the scope of this thesis.

2.1.2 Applications

As mentioned before, cryptographic hash functions find their place in many security applications, most notably in authentications related area such as digital signatures and message authentication codes. In this section some specific applications in which a hash function will be used will be given.

2.1.2.1 Password Protection

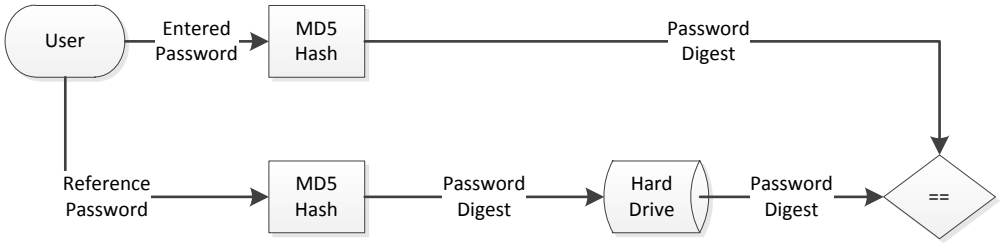

Figure 2.2: Password protection scheme of a CentOS system

One common place to use cryptographic hash is password storage. Almost all modern operating systems in the world require user authentication. The user will have to initially setup a password that the computer stores, in some form onto the hard drive. The next time someone tries to access that account, the stored password will be retrieved and compared to the password entered by the user.

This may pose a security problem because if the password is stored in plaintext in a storage device such as the hard-drive, someone may remove the storage device and extract physical data from it to obtain the password of that user. To prevent this, a cryptographic hash function may be used. Instead of storing the password directly into the hard drive, the password can run through a hash function first, and having the digest to be stored instead. When an authorized user tries to access that account later, his or her entry will be hashed and compared to the stored digest. Due to the deterministic nature of the hash function, if the user had entered his or her password correctly the digests will be a match and the user will be granted

access to that account. Throughout the process, the original password is never stored on the hard-drive but rather in its encrypted form. Figure 2.2 presents this scheme.

Several properties mentioned before play a critical role in the successful implementation of this scheme. Other than the obvious property that the hash function must be implementable, the one-way property and collision resistance help ensure that the unauthorized users cannot access the account. The one-way property guarantees that the unauthorized user cannon find the original password if he or she managed to extract the reference hash digest from the hard drive. A different message whose hash digest is the same as the original password is difficult to compute due to the weak collision resistance property.

This scheme for password security is used in Community Enterprise or CentOS, which uses the MD5 hash as the cryptographic hash function.

2.1.2.2 Data Integrity Verification

Another application in which cryptographic hash is commonly used is data integrity verification. This is to ensure that the data of interest is not altered during transmission or through tempering. In this case a message and its digest are given, and if the hash function is weak-collision resistant, it should be difficult to find a different message, either through randomly flipping bits or systematically replacing blocks of the original message. Virginia Tech software distribution system actually provides the digest of the md5sum, a program that computes the 128-bit MD5 hash, to help students to verify the integrity of the large files. This scheme can be easily mapped to Figure 2.2. In this case, however, instead of the user's password, the data files are of interest.

Therefore, the cryptographic hash can be used as a method for message authentication, wherein the hash digest can be used to prove the message has not been modified. The next point will be explained through a security problem with regards to a cryptographic hash to provide justification for the property of strong collision resistance. Since the cryptographic hash digest can be used as a proof of authenticity, an attack known as the birthday attack can be implemented this way. A person A, or Alice, prepares *m* number of

real contracts and *m* number of fraudulent contracts. The *m* number of real contracts have the same meaning but are written in different ways, such as substitutions of synonyms, adding or removing optional words or punctuations, etc. The *m* number of fraudulent contracts are prepared the same way. Then Alice will find a pair of contracts, one being real and the other one being fraudulent that have the same hash digest. After that, Alice will ask person B, or Bob, to sign the real contract but later claim that Bob has signed a different contract by providing the hash digest of a fraudulent contract. If the hash function is strong-collision resistant, which means it is difficult for Alice to come up with any pair of different messages that have the same hash digest, then *m*, the number of contracts *m* that Alice will have to try will be enormous [9]. Figure 2.3 shows the difference between weak collision and strong collision.

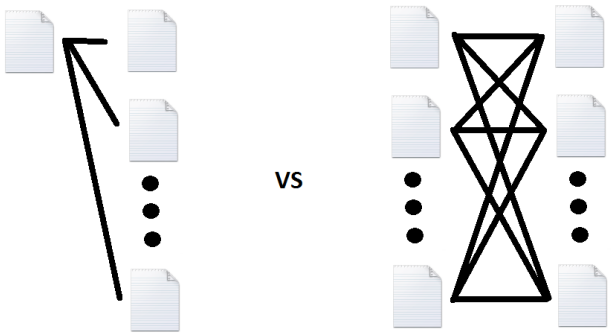

Figure 2.3: Weak collision and strong collision

In conclusion of this section, cryptographic hash is used in many other areas such as pseudorandom generation or key derivation, but the details of these applications will not be covered. Those who are interested can refer to any textbook or website on computer and network security.

2.2 What is SHA?

SHA stands for Secure Hashing Algorithm. It is a group of hash functions published by the National Institute of Standards and Technology as a US Federal Information Standard. All of the current SHA algorithms are developed by the NSA [1].

SHA-0: A 160-bit hash function published in 1993. It was quickly withdrawn due to an undisclosed flaw. It was replaced by SHA-1.

SHA-1: A 160-bit hash function that is similar to the earlier MD5 algorithm but more conservative. It is developed by the National Security Agency to be a part of the Digital Signature Algorithm. It is the most widely used SHA algorithm.

SHA-2: A family of two similar hash functions. It comes with four different sizes for the output, 224, 256, 384, and 512-bit. The 224-bit and 384-bit versions of SHA-2 are simply the 256-bit and 512-bit versions with truncated outputs.

SHA-3: This future hashing function is still under development. The algorithm will be developed by choosing different algorithms to a public competition. The final decision is expected to be announced in 2012.

2.3 Why SHA-3?

In 2005, Prof. Wang was able to use a differential attack on the SHA-1 hash function to find a hash collision more than one-hundred and thirty thousand times faster than what was acceptable [10]. Subsequent publications by other cryptanalysts were able to find hash collisions even faster. Despite the fact that those attacks cannot be extend to SHA-2, NIST decided that it was prudent to develop one or more hash functions as the transition from SHA-1 to the approved SHA-2 family is made.

NIST chose to develop the new hashing algorithm through a public competition, much like the competition in which AES was developed. The competition will accept algorithms from any person or organization [11]. The initial information regarding the competition, its minimum requirements and its evaluation criteria, was published in the first quarter of 2007. By the end of 2008, almost two years after the initial competition was announced, all the candidates had been submitted to NIST for evaluation. 64 algorithms were submitted, among which 51 submissions qualified for the first round.

In the second quarter of 2009, the First Hash Function Candidate Conference was hosted by NIST and the candidates that advanced to the second round were announced. According to the Round 1 report written by NIST, three broad categories of the evaluation criteria were used to evaluate the first round candidates: security, cost and performance, and algorithm and implementation characteristics in software. The candidates' performance in hardware is not evaluated at this stage because the implementation of the candidates in hardware requires hardware designers to rewrite the algorithms in HDL, which is too extensive for any organization considering the number of available candidates.

In the second quarter of 2010, a year after the First Hash Function Candidate Conference, the Second Hash Function Candidate Conference was held by NIST and the winners of the second round were announced. Five candidates were selected among the 14 competitors to advance into the third round. NIST still adhered to the three categories of evaluation criteria from round 1 during the round 2 process. However, the evaluation criteria had been extended to the hardware domain since a few organizations were able to implement all the remaining candidates in hardware, thus making comparisons possible between the candidates.

Finally the winner of the competition will be announced sometime in 2012 [12]. The winner of the competition will be crowned with the title "SHA-3," and the evaluation of hardware performance of the candidates in the competition is this project. The hardware performance evaluation of the round three candidates is covered in chapter 3.

2.4 Common Hashing Algorithm Components

According to the famous information theorist Claude Shannon, two primitive operations are needed to build strong encryption algorithms: confusion and diffusion. Confusion is the encryption operation to make the relationship between the message and its digest obscure. Diffusion on the other hand is an encryption operation to spread the influence of each message bit so that its statistical property is hidden

[13]. Clearly, the confusion operation helps to maintain the one-way property and the diffusion strengthens collision resistance of the hashing functions.

Even though the individual hashing candidates differ greatly, they all use common components to carry out the operations mentioned above. Some of these components and their typical implementation characteristics in software and hardware are described below.

2.4.1 Permutation

A permutation operation is primarily swapping of data. The swapping of data carries out the diffusion operation due to the data being swapped in different places.

The size of the data that is being swapped depends on the algorithm. The ones that swap at a smaller scale can swap data in bits, while the ones at a larger scale can swap multiple words.

In software, permutation operations are assignment operations which involve moving data in memory. However, when the data swapping must occur at a scale less than 8 bits, logical masking must be implemented. In hardware, the permutation operations are very efficient since only wiring is involved, thus the resource cost is at a minimum [14].

2.4.2 Substitution

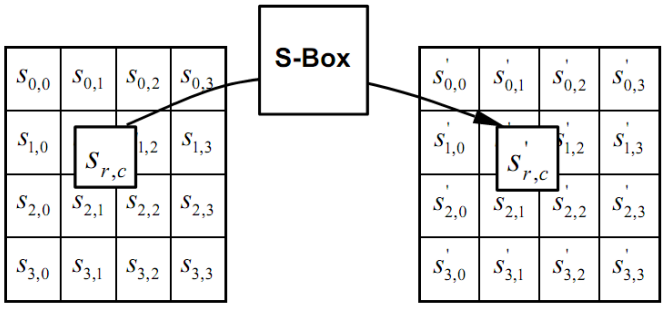

Figure 2.4: An example of an S-Box

A substitution operation is a non-linear transformation of the input. It is a common operation that is used to carry out the confusion operation. Typically, the substitution operation is implemented using a substitution-box, or S-Box. The S-Boxes are carefully chosen to be resistant to cryptanalysis and they are built in different sizes. Figure 2.4 shows an example of an S-box.

In software, the S-Boxes are typically implemented as two dimensional constant arrays. Due to the sequential nature of a software program, only one instance of the S-Box needs to be declared regardless of the frequency of access to the S-Box the program makes. In hardware on the other hand, the S-Boxes are typically implemented as lookup tables, so multiple instantiations must be made if parallel access is desired, increasing the resource cost.

2.4.3 Logical function

Logical functions are performed using logical gates such as AND, OR and NOT gates. The most commonly used logic gate is the XOR. The XOR gate is a desirable function in cryptography because it has a balanced function. Therefore it is impossible to know the inputs to an XOR gate by looking at the output alone.

The logical functions are implemented in similar ways in software and in hardware. In software, they are implemented via logical operations. In hardware, they are implemented via logical gates.

2.4.4 Modular arithmetic function

Modular arithmetic functions are typically used for the purpose of diffusion. Two commonly used modular arithmetic functions are modular addition and multiplication. These are usually used along with shifting. The diffusion operation is carried out due to the generation and the propagation of the carry within the arithmetic operation.

In software, they are implemented with arithmetic operations. In hardware, they are implemented through a modular adder or multiplier. Depending on the size of the operands, the hardware resources required

may be large especially for the multiplier. In addition, due to the carry operations, the modular arithmetic functions are typically slower.

2.5 Round-3 candidate hardware structures

In this section, the structures of the 3rd round candidates implemented by this group are presented.

2.5.1 Blake

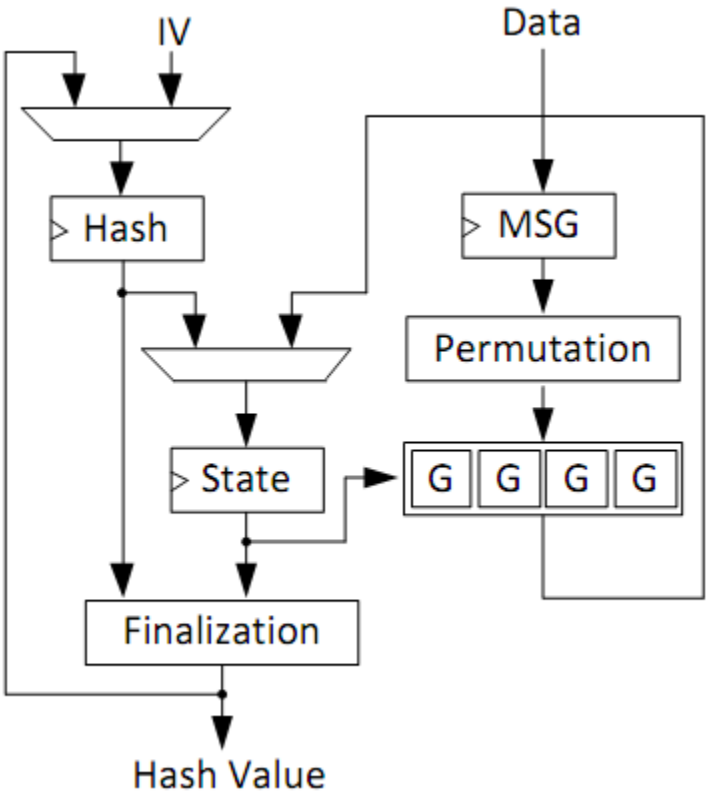

Figure 2.5: Structure of the Blake-256 core

Blake implements the Hash Iterative Framework (HAIFA) construction for its iteration mode. The HAIFA construction produces digests based not only upon the input message and internal state, such as the case of the more commonly used Merkle-Damgard construction, but also on the number of processed bits recorded by a counter.

The internal states of Blake will first be initialized using a set of initial values, counter values, and some constants. The round function of Blake is based on the ChaCha stream cipher. It is composed of two layers of G functions in parallel with four G functions in each layer. A G function uses modular addition, XOR, and rotational shifts.

As shown in Figure 2.5, the input message data first goes through a permutation process and then it is sent to one layer of G functions along with the stored internal states. Since one round requires two layers of G functions, each cycle only completes half a round. The output of the half round function will be stored inside the internal state registers for the next half round. After the completion of rounds sequence for the current block, a finalization process is taken and the chaining value for the next input message block is stored in the internal state registers. In the design made by this group, the layer of G functions can be unrolled into two layers to decrease the latency at the expense of maximum frequency and area.

2.5.2 Groestl

Groestl implements a wide-pipe Merkle-Damgard construction with logic operation and truncation at the final output. A wide-pipe Merkle-Damgard construction is similar to the plain Merkle-Damgard construction but the size of the wide-pipe's internal state registers is bigger than the size of the output digest.

Groestl mainly has two functions, P and Q, which resemble the Rijndael block cipher. Both functions include four round transformation functions, which mainly performs XOR, permutation, and substitution. Then the outputs of P and Q will be used as the inputs of XOR gates and the outputs of which will be stored as the hash digest.

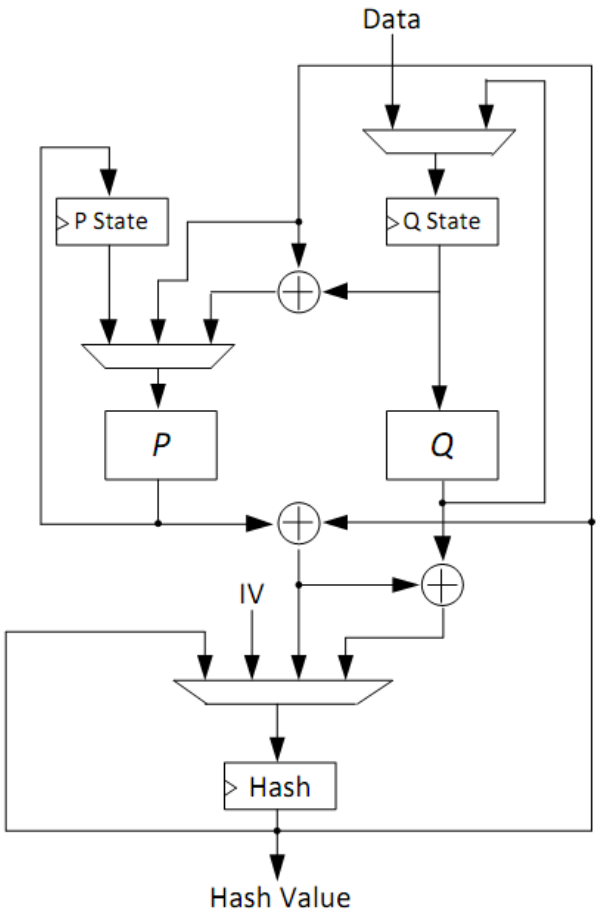

Figure 2.6: Structure of the Groestl-256 core

As shown in Figure 2.6 the implementation functions P and Q are implemented in parallel. However, due to their structural similarity, their hardware resources can be shared. A low area implementation of Groestl can be sought by combining the P and Q functions to save hardware resources.

Within the P and Q structure, 128 AES S-boxes are implemented in Galois field operations. They may be implemented as lookup-tables which may improve the performance on FPGAs.

2.5.3 JH

The compression function of JH is comprised of permutation, substitution, and logical XOR. Before each round function, the first half of the chaining values are XORed with the input message. Then at the end of each round function, the second half of the chaining values are XORed with the input message.

As shown in Figure 2.7, the main logic of JH is the module R6 and R8. R6 is a round constant generator. In each cycle, the round constants are generated based on either the round constants from the previous cycle or the initial constant values. The R8 module generates the new chaining values based on the input message, chaining values from the previous cycle, and generated round constants. A layer of substitution with the S-boxes, a layer of linear transformation, and a layer of permutation are presented inside R8.

Some potential tweaks are available for JH. First the S-Boxes can be implemented with a set of logical functions instead of lookup-tables. Second, the round constant generator, R6, can be replaced with a ROM since all the possible round constant values can be computed beforehand.

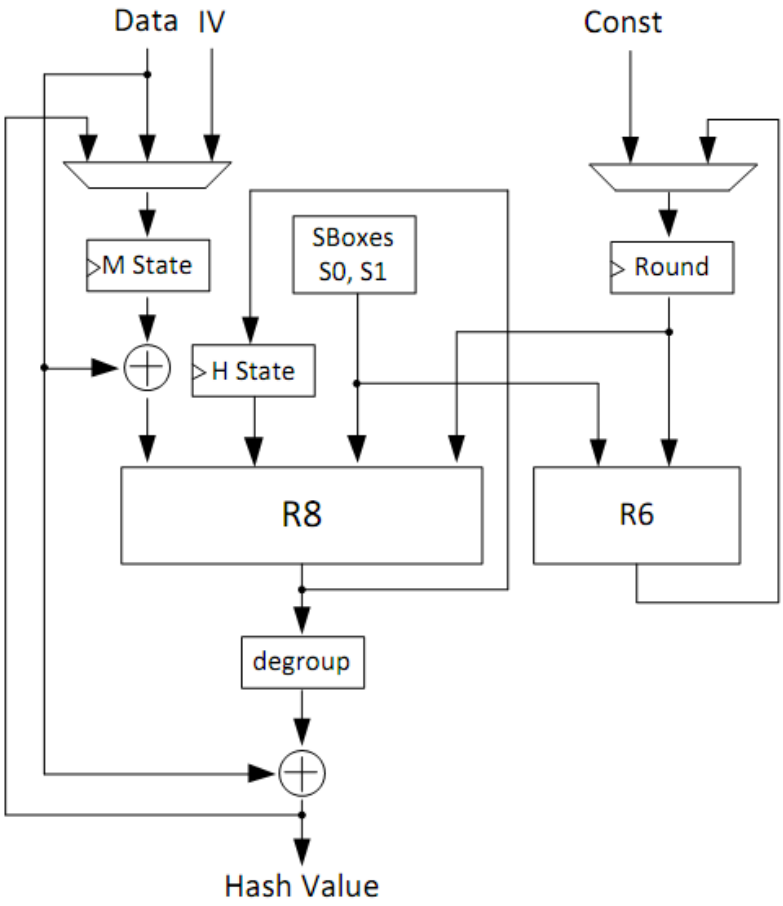

Figure 2.7: Structure of the JH-256 core

2.5.4 Keccak

Keccak uses the sponge construction model. In a sponge construction, the internal stage registers are divided into two portions. A sponge construction has two phases, absorb and squeeze. During the absorption phase, the input message will be XORed with the data stored inside the first portion. The XORed value will be updated along with the data from the second portion of the internal state registers through the round function. During the squeeze phase, the data stored inside the first portion will be used as a portion of the output. New output values will be present inside those registers as they were updated through the round function in each cycle. Unlike the structures mentioned previously which truncate the internal state registers for output, a sponge construction can accommodate an output of any arbitrary size by updating its internal state registers. Figure 2.8 shows the overview of the Keccak-256 core.

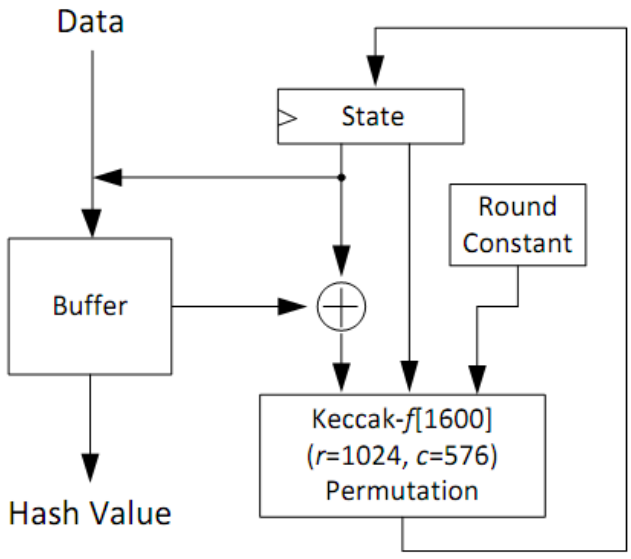

Figure 2.8: Structure of the Keccak-256 core

The Keccak has a round function module and a round constant generation module. The round function module of Keccak is a permutation-substitution network using S-Boxes that are 5-bits wide. The round generation module generates round constants based on a 5-bit round number.

The implemented Keccak has 1600 internal state registers. The bitrate of the implementation has the first 2013 internal state registers and the s576 remaining registers are known as the capacity of the implementation. This design is based on the Keccak author's reference for high speed core implementation.

2.5.5 Skein

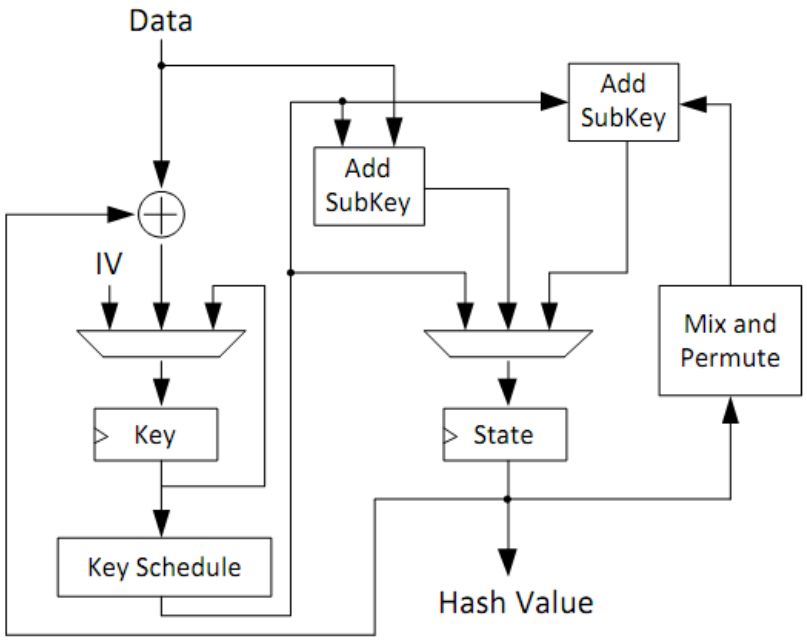

Figure 2.9: Structure of the Skein-256 core

Skein is an iterative hash algorithm built on a tunable block cipher called Threefish. The Threefish block cipher is used to build the compression function of Skein using a modified Mateas-Meyer-Oseas configuration. The Threefish is primarily composed of subkey, mixing, and permutation. The subkey operation is composed of three modular additions. The mixing operation involves modular addition, rotational shift, and XOR.

Figure 2.9 shows Skein512-256's implementation. Each Threefish-512 round out of a total of 72 rounds consists of four mix operations in parallel and a permutation. Four rounds of the design have been unrolled. Thus, a total of 16 mix functions are present in the design.

2.6 Hardware Devices

In this section, the implementation characteristics and development procedure of two hardware devices will be discussed.

2.6.1 Application Specific Integrated Circuit

2.6.1.1 Characteristics

Logic functions of ASIC devices are carried out by logic cells, to which the HDL design will be mapped. The designers can make custom cells or use the standard cells from a library. The placement and routing of the cells is typically done automatically by the ASIC designing tools. After the design process is finished, a layout mask is generated by those ASIC design tools and sent to the fabrication plants to fabricate. To create one of those layout masks is very costly. If a mistake is made or a change is needed and there are no configurable portions on the chip that can perform the desirable change, a new layout mask must be made.

However, the design space of an ASIC implementation is a lot greater; the designer has many possible adjustments can be made in the ASIC design process. In addition, an implementation from the same HDL source is typically many times smaller, faster and consumes less power when they are implemented in ASIC rather than on an FPGA.

2.6.1.2 Design Process

The design process of an ASIC chip is much more cumbersome than the design process of FPGA. After the HDL is developed and the behavior simulation is completed, synthesis, floor-planning, automatic place and route would have to be performed, which takes at least a few weeks. Many manual fixes, tweaks, and verification steps are taken along the way to ensure the desired results are given. After the final stage of the process, a layout mask is produced and sent to the fabrication plant. Then, in about one to three months the fabricated chip is returned.

2.6.2 Field Programmable Gate Array

2.6.2.1 Characteristics

The architecture of an FPGA implementation is drastically different from its ASIC counterpart. An implementation of an ASIC device involves using different logic cells to build an integrated circuit that yields the desirable behavior. An FPGA implementation, on the other hand, involves breaking down the design and map into existing logic cells. The primitive logic of FPGA devices is a lookup table, or a LUT. The size of LUTs can vary depend upon the device family. The LUTs can be perceived as truth tables; therefore, a LUT with *n* number of inputs denoted as n-LUT can implement any combinational logic that has *n* number of inputs or less. An FPGA logic cell is typically comprised of one or more LUTs, some multiplexers, and registers. The wiring of the logical cells is done through programmable interconnects.

One big advantage of an FPGA is that it can be changed after manufacturing, some even during operation. Therefore, the cost of changing, upgrading, or correcting an existing FPGA implementation is minimal compared to an ASIC device due to their device flexibility.

2.6.2.2 Design Process

The design process of an FPGA implementation is more straightforward. In most scenarios, the entire design process is carried out by software distributed by the manufactures of the FPGA, with minimal user constraints and configuration. The time between finishing a working RTL and a working FPGA chip is typically less than a few hours.

2.7 Conclusion

This chapter established some background information regarding to cryptographic hash, such as its description, applications, and properties. Then a description of the SHA family and how SHA-3 came into play was provided. After that the commonly used operations and modules used by cryptographic hash

functions were discussed. In the end the five SHA-3 candidates and the hardware devices where those candidates will be implemented were described.

Chapter 3

Hardware Performance Analysis

In this chapter, the detailed information regarding the experiments will be discussed. The interface chosen, reasons behind the selection of a particular technology node, CAD flow for both ASIC and FPGA will be explained.

3.1 Methodology

3.1.1 Interface

A variety of hardware interface schemes were proposed by various groups in [15] [16] [17]. A common interface among all candidates is a must for fair comparison. For this work, the interface proposal by Chen et al[19] was chosen. The interface includes two signals to set the hardware back to the initial state: reset and init, reset, which is asynchronous, and init which is synchronous. It also includes a 16 bit bus I/O for data transfer and a load, fetch and acknowledge signal for control. It should be noted that other proposals may be valid as well, and those who are interested can consult [18] for more comprehensive comparison between different interfaces.

3.1.2 Metrics

To compare different designs, metrics are required for the purpose of evaluation. The candidates are characterized by the following five primary metrics.

Area: The size of the design. For ASICs, it is typically expressed in terms of gate equivalent or GE. GE is calculated by the area of the design divided by the area of a two input nand gate from the same library. The area of a design on an FPGA is calculated in terms of the number of logic cells it occupies. Since all of the FPGAs that the designs will be mapped to are Xilinx FPGAs, the basic logic cell is known to be a slice and the number of slices the design is using is given by the Xilinx tool-set. The area metric is

one of the more important metrics which many other groups that are working on SHA-3 focus on. It is a major indicator of the resource cost of a design.

Maximum Frequency: For ASIC, the maximum frequency was obtained at the post-layout stage using Synopsys PrimeTime. At post-layout stage PrimeTime will take cell delay, wire delay and cell parasitics into account. For FPGA, the timing information is extracted from the Xilinx TRACE reports.

Maximum Throughput: the throughput of a design is calculated as follows

$$TP = \frac{BlockSize * MaxFreq}{Latency}$$

The block size is the amount of data the hashing algorithm will process at a time. It is fixed by the algorithm and is not a design parameter for the hardware designer. The latency shown here is the core latency, which is the number of cycles it takes to hash a message. The maximum throughput is another important metric people focus on in the SHA-3 project because it indicates how much data a design can process over a fix amount of time. Maximum throughput is important because this number weighs block size, a characteristic of the algorithm; maximum frequency, a characteristic of the hardware design performance, and latency; a characteristic of the hardware design architecture.

Power: The power of a design is constituted by active power and static power. The active power of the design is due to the circuit switching. The static power is due to the leakage of the logic cells. For ASICs, the power of each design is collect using Synopsys PrimeTime after backward annotation using a Standard Parasitic Exchange Format file and a Value Change Dump file which provide the parasitics and switching activity of the circuit. The netlist used is the post-layout netlist. For FPGAs, the power of each design is collected using Xilinx XPower tool. The hash modules were running at 25 MHz on ASIC, and 20 MHz on FPGA.

Energy: The energy of a design is expressed as energy per bit of the input block message, which computed using the equation $E = \frac{T*P_{total}*Latency}{BlockSize}$, where T is the period of the clock, P is the total power dissipation.

These five metrics shown above are most often used when presenting hardware performance results of these algorithms. In the experiments, maximizing the throughput per area ratio, which is calculated by dividing the maximum throughput by the area of the design, is the optimization goal. This parameter was chosen because of its comprehensive coverage of both performance and cost of the algorithm and hardware design.

3.1.3 Technology Node Selection

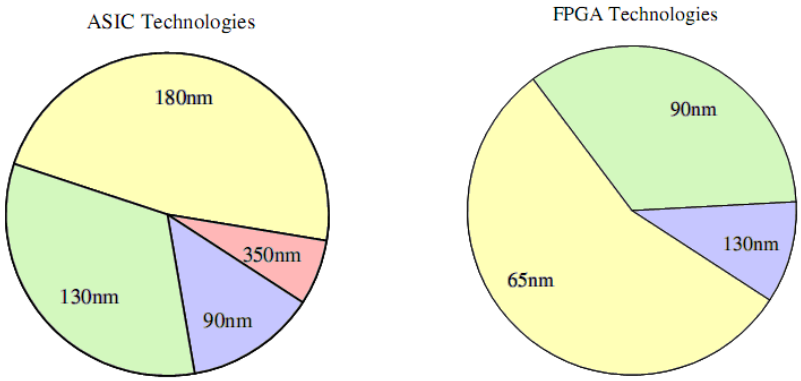

Figure 3.1: Technology nodes used for ASIC and FPGA hash implementations within the last 5 years

Many analyses that cover hardware performance of the SHA-3 algorithm in FPGA, ASIC or both use various technology nodes for both domains. During an introductory survey of hash hardware implementation papers published in CHES, Cryptography ePrint and SHA-3 zoo from 2005, it was shown that, for 90 reported hash implementations in FPGA, 56% of them use 65nm FPGAs and 34% use 90nm FPGAs. For 61 ASIC implementations, on the other hand, 48% of the designs chose 180nm while 33% chose 130nm. Figure 3.1 shows the pie charts from the studies. It appears that the most popular ASIC technology is several generations behind FPGAs, from 180nm to 65nm. Except for high-end hardware components such as microprocessors, similar trends exist on other industry designed hardware. The

130nm technology node was chosen for ASIC and for FPGA the 65 nm technology node was chosen for the comparisons. The data for four families of FPGAs in 90, 65 and 40nm technology nodes to cover both high and low-end FPGA devices will be provided, but the cross-device or cross-platform comparisons will be focusing on the 65nm FPGA.

The impact of technology scaling on both ASICs and FPGAs was also evaluated. For FPGAs, the scaling factors are generally hard to quantify because different FPGA families may have drastically different architectures. In [19], researchers have already demonstrated the influence of different technology nodes on the FPGA results for SHA-3 Round 2 candidates. For example, when moving from a Xilinx Spartan3E to a Xilinx Virtex-5, not only did the feature size change from 90nm to 65nm, but the basic logic element also changes from 4-LUT to 6-LUT. In addition, the presence of hardened IP blocks, such as embedded memory (Block RAM), clocking management blocks and DSP functions, can lead to differences between two FPGAs even within the same technology node. Therefore, to avoid further complications, the comparisons of the SHA-3 candidates on FPGAs will only use mapped netlist that don't infer any hardened IP blocks.

For ASICs, the scaling factor across different feature sizes is expected to be more linear. A preliminary study on the SHA-3 candidates to evaluate the impact of different technology nodes (90nm vs. 130nm standard cell ASICs) was done, the details of which will be shown in the next chapter.

3.2 CAD flow

3.2.1 CAD flow for FPGA

For hardware performance evaluation on FPGA, all of the five SHA-3 designs in HDLs are implemented on Xilinx Spartan 3, Virtex 4, Virtex 5 and Virtex 6. Two approaches were used for FPGA CAD flow, the Xilinx Xflow and the ATHENA flow. In both of these flows, the usage for hardware macros had been turned off to decrease the complexity of the comparison. Both of these flows follow Xilinx synthesis, mapping, place and route, and timing analysis process.

3.2.1.1 Xilinx Xflow

Xilinx Xflow is a command line program that automates Xilinx synthesis, implementation and simulation flows. Xilinx Xflow binds synthesis, mapping, place and route into one command. Once the design files and the device model are specified, Xflow can use its own set of default options if custom configurations are not given. In the experiments, the default Xflow options are chosen, with no placement or frequency constraints given. The results obtained from this flow can be considered as being the typical case.

3.2.1.2 ATHENA flow

The Automated Tool for Hardware Evaluation (ATHENA) is a hardware benchmarking tool targeting FPGAs created by a group in GMU. The ATHENA software still uses the software provided by Xilinx for all the design stages, however, ATHENA can automatically tweak and tune options and switches as it sees appropriate to yield the best possible outcome for a design. According to [20], the performance of a FPGA implementation is dependent upon the constraints and the value set for the cost table. The cost table is a seed to randomize the place and route process and using different values for cost table can yield different results. A total of one hundred different cost table values is possible.

Figure 3.2, 3.3 and 3.4 are made from data provided in [20]. In this study, a design was run through the Xilinx FPGA implementation process with all one hundred different values for the cost table. Knowing the design is able to run at 90 MHz, the frequency constraint was set under, at and over that frequency. Figure 3.2 shows the resulting frequency of the design when it is under-constrained. It is evident that the resulting design has met the constraint fewer than sixty percent of the time and none of the implementations of the design were able to achieve 90 MHz as their maximum frequency. Figure 3.3 shows the results of the design when the constraint is set at the maximum possible frequency. The resulting design has met the constraint four out of one hundred times. Figure 3.4 shows the results of the design when it is the constrained above its maximum frequency. None of the resulting implementations met the constraint and none of them were able to run at 90 MHz.

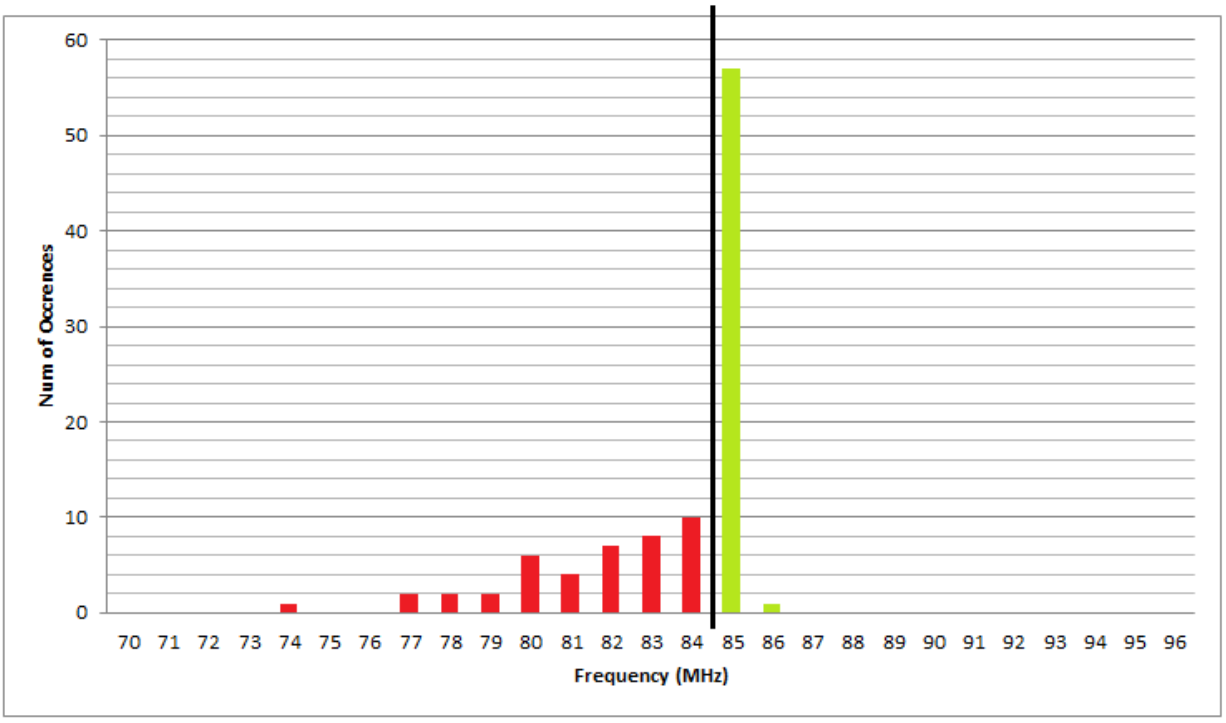

Figure 3.2: Distribution of the actual clock frequencies that the design can achieve with the constraint being set at 85 MHz from [20].

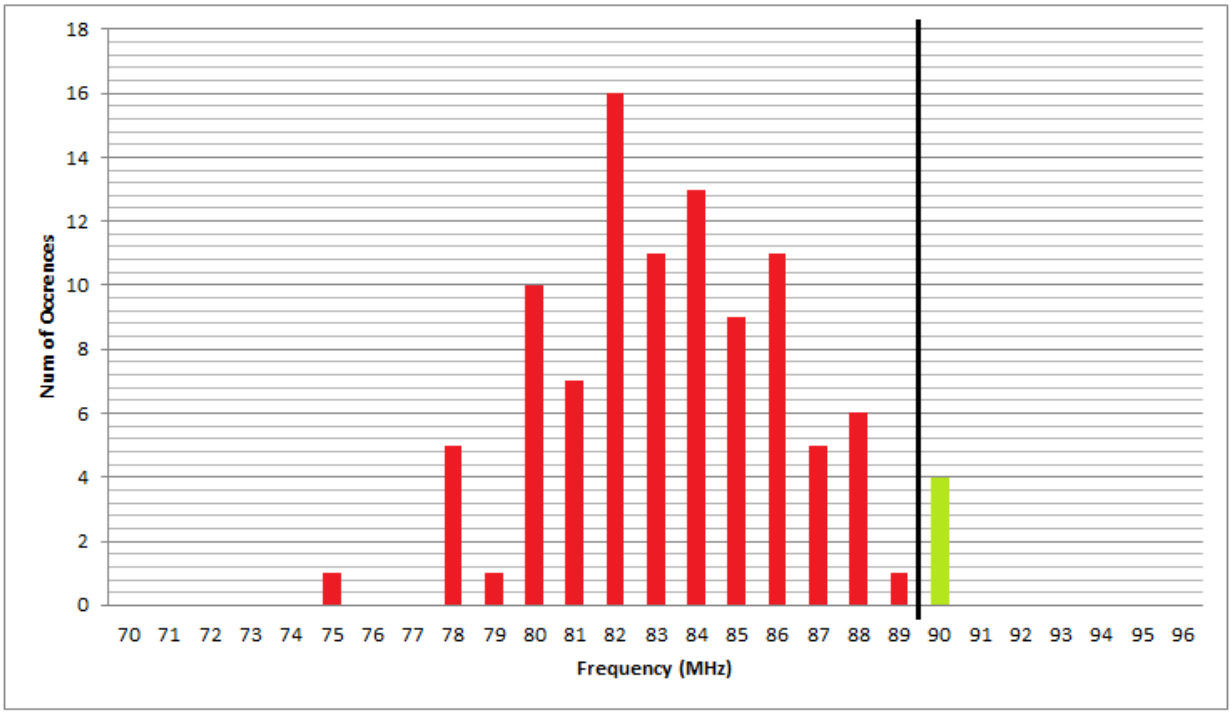

Figure 3.3: Distribution of the actual clock frequencies that the design can achieve with the constraint being set at 90 MHz [20].

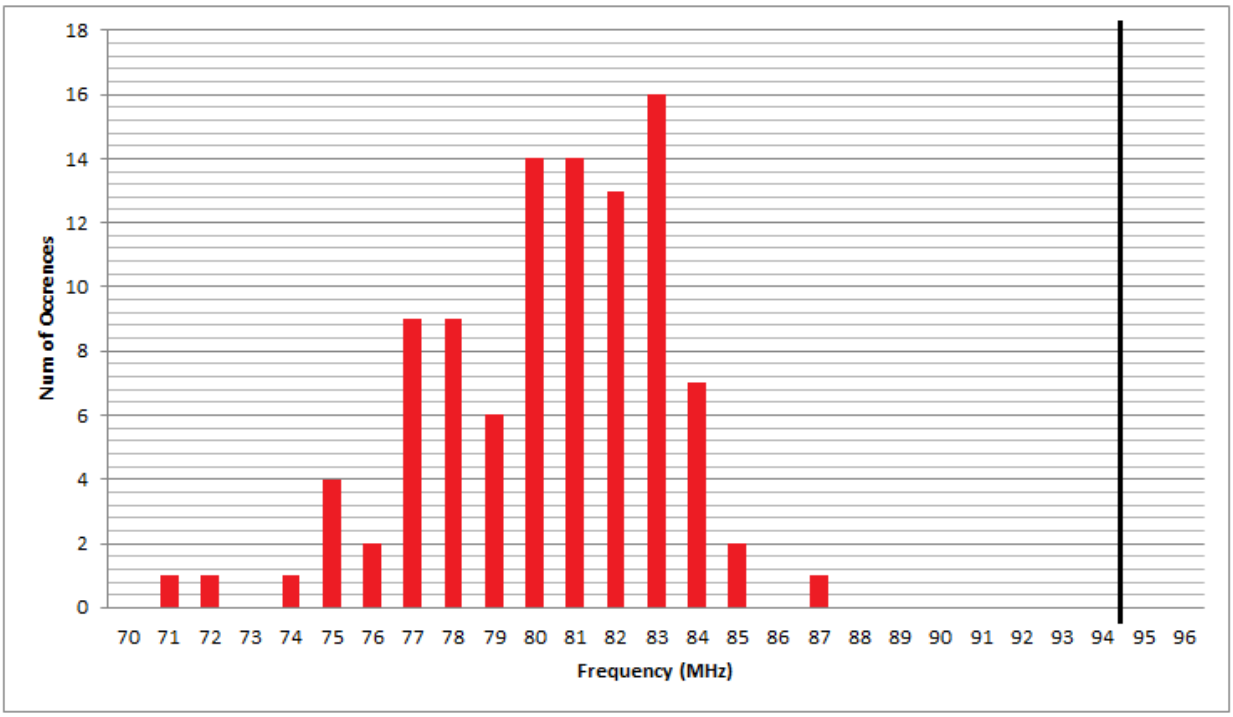

Figure 3.4: Distribution of the actual clock frequencies that the design can achieve with the constraint being set 95 MHz [20].

Therefore, due to the sensitivity the Xilinx software has to placement and frequency constraints, under or over constraining the design may give sub-optimal results. This is the motivation to use the ATHENA flow, as it gives a more accurate representation of the performance of the candidate designs.

Many options are present in the ATHENA program that allow the user to configure the granularity of the tweaking process. The granularity that was chosen will let the software search through some values for the cost table and tune the frequency constraint to give an acceptable evaluation while finishing the process within a reasonable amount of time.

The reason for presenting the Xflow results on the other hand, is to give a measure of how well the candidates score on “first impression”, since default settings are typically used when implementing a design on a FPGA. In addition, design exploration tools like ATHENA aren’t very well known out-side of this project.

3.2.2 CAD flow for ASIC

Hardware performance evaluation on ASICs on the other hand, is a lot less straightforward. Figure 3.5 shows the overview of the ASIC design flow. First, the Synopsys Design Compiler was used to map the RTL codes into primitive gates. The typical case with nominal voltage, process and temperature was chosen when characterizing the condition of the standard cells.

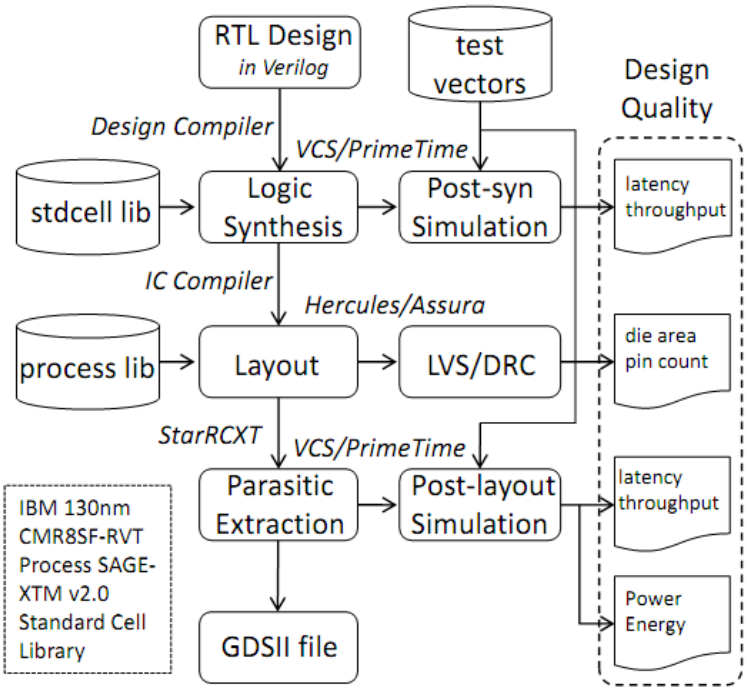

Figure 3.5: The overview of the ASIC design flow

As is the case with FPGA CAD flow, even with the same RTL source code, different constraints and optimization goals can lead to design outputs that are drastically different in terms of performance and area. Four design points for each implementation were evaluated:

MinArea: At this design point, all of the optimization effort will be directed on minimizing the overall area of the design, thus smaller gates with less drive strength will be used. The design is expected to be the slowest at this design point.

MaxSpeed: All the optimization effort will be spent on increasing the maximum frequency. Gates with greater drive strength will be used, and these gates are typically bigger in size. In this design point the overall area of the design is expected to be a lot larger than the design at the MinArea point.

TradeOff0: When attempting to optimize for area or performance, the law of diminishing marginal return applies to ASIC design. In other words, at the two extreme design points, MinArea and MaxSpeed, one parameter was significantly over-paid to gain marginal improvement for the other. A possible cause is that the design may already be saturated in terms of speed or area, but due to the optimization constraints set by the user, the tool continues to push for faster speed or smaller area while significantly sacrificing the other parameter although the actual gain for the optimization goal is very small. For example, at the design point of MaxSpeed, some fast logical cells could be exchanged for slower but smaller cells which will slightly decrease in circuit speed while reducing the overall area of the design by a significant amount. This will help increase the maximum throughput per area ratio. This trade-off point is constrained to be two–thirds between the MinArea and MaxSpeed.

TradeOff1: Likewise, this design point is another choice picked. It is constrained at five-sixth between the MinArea and MaxSpeed.

The trade-off points were investigated to characterize how the relationship between speed and area evolves as one of them is being optimized.

The Synopsys IC compiler is used for the backend process, such as floor-planning, placement, signal tree synthesis, parasitics extraction, and routing. In the end, Hercules and Assura were used to perform DRC error checking. VCS was used for circuit simulation and verification.

3.2.3 Design Strategy

Three types of implementation schemes exist, namely fully autonomous, external memory and core functionality [21]. A brief description is given below.

Fully autonomous implementation includes the full functionality of the candidate algorithm. The input message can be loaded piece by piece into the hardware module, and the hardware digest will appear at the output. For such implementation scheme, the throughput of the module will depend on the hash structure within the hardware and how fast the data can be loaded into and out from that piece of hardware. Figure 3.6a shows this strategy.

Implementation with external memory relies on an external memory to hold intermediate results or even the entire input message. It typically includes the logical functions of the hash algorithm, some registers for small temporary values and potentially some memory controller module for interfacing between the core logic and external memory. If the input message isn't stored in the external memory, then the implementation must receive the input message similar to the way a fully autonomous implementation receives the input message. For such an implementation scheme, the speed of the external memory can potentially be the bottleneck of the performance. Figure 3.6b shows this strategy.

Finally, the implementation of core functionality only includes the core logic of the hash function, such as the compression function. It mainly serves the purpose of giving quick and rough estimates regarding to the performance of a design. Figure 3.6c shows this strategy.

All of the candidates implemented are fully autonomous and can be used as hardware IP on a chip.

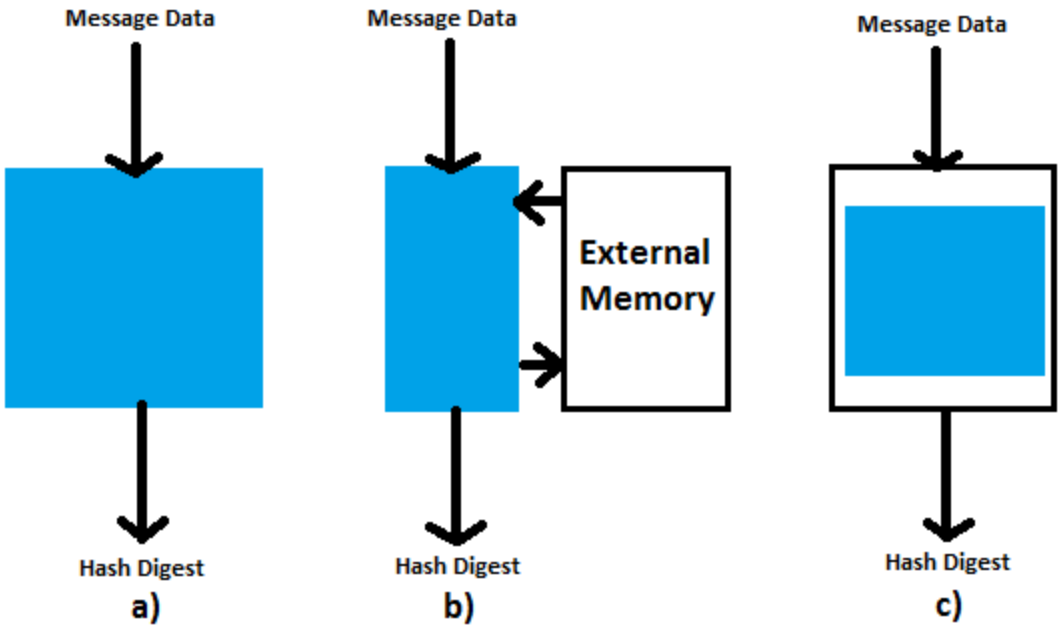

Figure 3.6: Three different design strategies. a) shows the fully autonomous strategy. b) shows the design with external memory. c) shows the design with core functionality

3.3 Framework

3.3.1 Overview of the SHA-3 Evaluation project

Involvement with the SHA-3 project started after NIST announced the 14 candidates that made it through round-one in the first SHA-3 conference. It was then that the number of possible candidates became small enough for a thorough hardware performance evaluation of all candidates. This group at Virginia Tech was in an international collaboration among several research groups in developing the RTL designs of the 14 second round two candidates. From this collaboration, not only did this group receive all the RTL code needed to perform experiments, but also benefited from the suggestions made from other groups. Currently, all the designs in this project are all updated to their round-3 specification.

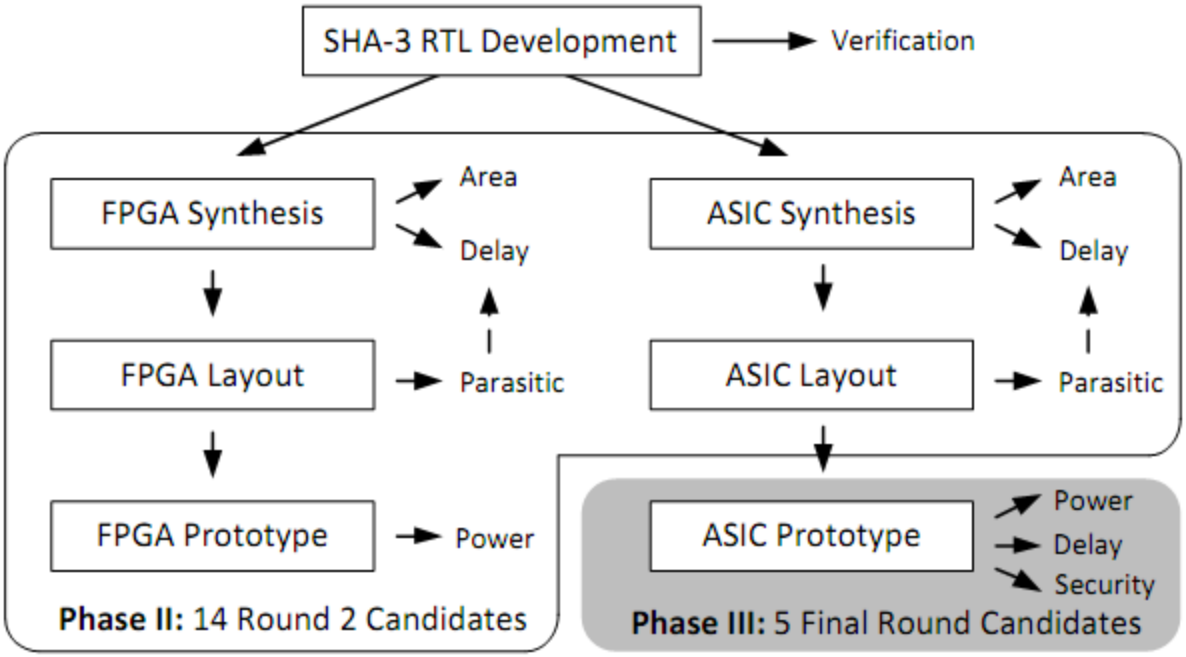

Figure 3.7: An overview of the SHA-3 ASIC evaluation project

Figure 3.7 illustrates the overview of the SHA-3 hardware evaluation project. In round-two, the RTL codes for the 14 round-two hash candidates were obtained through the collaboration. All fourteen round-two candidates had run through the ASIC and FPGA phase-two flow and were evaluated. However, the RTL for those candidates are no longer up-to-date. In addition, the ASIC CAD follows the Synopsys Reference Methodology. The data collected using this methodology may serve as good reference points; however, to bring out the full picture, data needs to be collected from a taped-out chip. In phase three of this project, all the designs have been sent for manufacture. The details of the ASIC implementation will be shown later in this chapter. The FGPA flow was also redone with Xflow and ATHENA flow mentioned above.

Finally, after the chip comes back, additional experiments and measurements of the designs can be performed on both FPGA and ASIC devices. In the next section, the FPGA prototyping and ASIC testing will be discussed.

3.3.2 Evaluation Platform

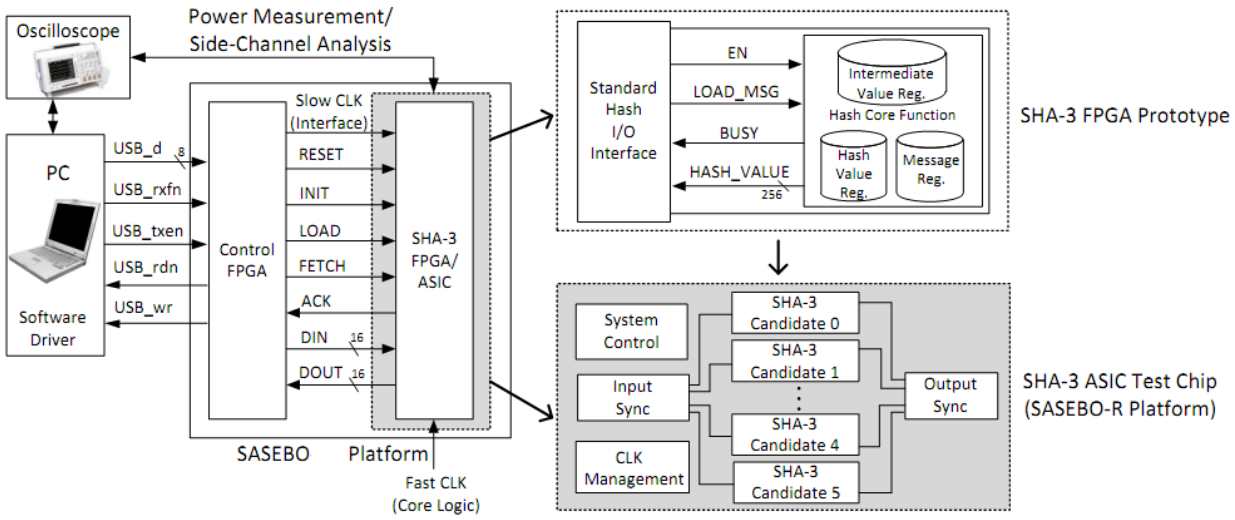

Figure 3.8: The experimental environment for FPGA prototyping and ASIC testing

In this section, the prototyping platforms for both FPGAs and ASICs will be introduced. The overall structures of the evaluation platforms for FPGA and ASIC devices are shown on Figure 3.8. It can be seen that they are very similar. Both evaluation platforms contain a PC, a SASEBO board and an oscilloscope.

The PC serves mainly for verification purpose and it will have the software implementations of the SHA-3 algorithms. It will generate a number of different messages on the PC that will be hashed by the SHA-3 algorithms. The PC sends these messages to the cryptographic hardware, which contains the hardware implementations of the SHA-3 algorithms. When the cryptographic hardware finishes hashing and returns the hash digest back to the PC, the PC can then compare the results between hardware and the software to verify correctness.

The SASEBO boards are chosen to be the cryptographic hardware evaluation platform. For FPGA prototyping, the SASEBO-GII is used. The SASEBO-GII has two FPGAs on board: a Spartan 3 and a Virtex 5. The Spartan 3 will be the control FPGA, which is used as the interface between the PC and the hardware implementation of the SHA-3 algorithms. The SHA-3 algorithms will be implemented on the Virtex 5. It will be referred to as the cryptographic FPGA. For ASIC prototyping, the SASEBO-R is used. The SASEBO-R has one FPGA and one ASIC chip slot. The FPGA on the SASEBO-R is a Spartan 3 and

will still be used as the control FPGA. The ASIC chip slot on the other hand, will hold the fabricated chip. The SASEBO boards were chosen because they were designed for cryptographic hardware evaluation, which makes future security and performance analysis easier.

3.4 ASIC Implementation

In this section, the ASIC implementation will be discussed. First, a brief description of the chip's architecture will be given. Then the constraints imposed on the designs will be presented. Finally, the design environment will be described.

3.4.1 Overall Chip Architecture

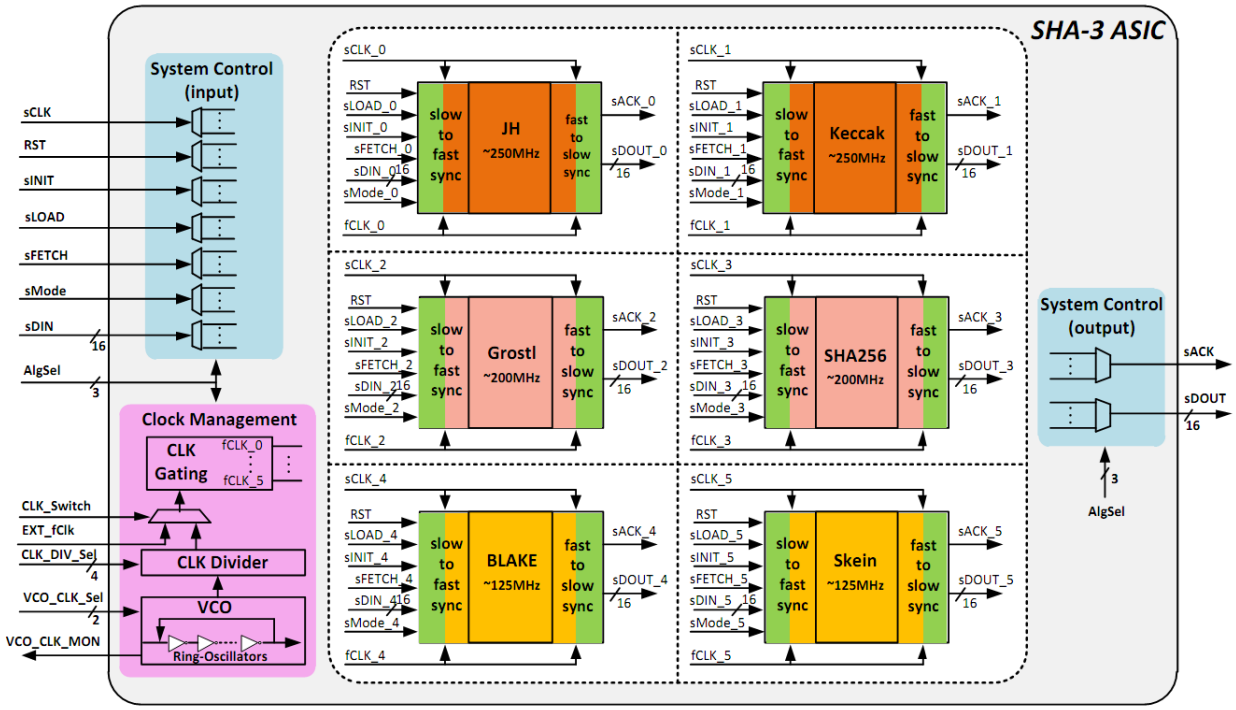

Figure 3.9: An overview of the SHA-3 ASIC chip implementation

The chip includes six hashing modules, a system control module and a clock management module. The six hashing modules include the five SHA-3 round-three candidates and a SHA-2 core. The SHA-2 module is included as a reference due to its simplicity. The SHA-2 is often used as a normalization factor for comparison across different experiments. Each of the six hashing modules includes a clock

synchronization module. The synchronization modules are included for the purpose of accommodating the two clock domains on the SASEBO-R board. Figure 3.9 shows an overview of the SHA-3 ASIC chip implementation.

3.4.2 Clock Management Module

The clock management module serves two purposes: on-chip clock generation and clock gating. The reason why an on-chip clock generator is built is because all the control signals are given by the on-board control FPGA. The control FPGA operates at low frequencies and sending high frequency signals using on-board connection results in heavy distortion. In addition, the switching capability of the input pad cell on the ASIC chip was not known. Therefore, the decision was made that a voltage control oscillator (VCO) to generate the clock. The custom designed VCO is controlled by an external analog voltage, which controls the oscillation frequency. The output of the VCO will be sent into a clock divider circuit. Multiple divided clock signals will be sent into a MUX and one of those signals will be selected. The selection for the clock will be done through the off-chip switches. An external off-chip clock can also be selected if the on-chip clock generator fails. These components can be seen on Figure 3.9.

The clock gating logic is implemented for the accurate power measurement. If the clock gating was not used, the collection of power data of a single module will be affected by other modules on the chip.

3.4.3 System Control Module

The system control module interfaces between the hashing modules and the control FPGA. It controls the input, output of the data, reset, initialization and mode selection. The module supports two modes: normal hashing mode and full speed testing mode. In the normal hashing mode, the input message will be sent piecewise into the hashing module. After the hashing module receives a full block of message data, it will start hashing, and returns the digest after the entire message is hashed. Under the full speed mode, the hashing module will only receive 16 bits of input message. Then, that piece of message is automatically replicated enough times to fill the input message block. Thus, after one cycle, the hashing module can

begin hashing. The reason behind the implementation of this mode is because, between each hash operation, the hashing module must wait at least 32 cycles for the new message block to be filled in. This creates a challenge if only the power consumption during hashing needs to be measured. The full speed testing mode will allow the hashing module to run continuously for a long time.

3.4.4 Clock Synchronization Module

Two clock domains are present on the chip: a slow clock domain for the interfacing modules and a fast clock domain for the hashing modules. To avoid complex synchronizers such as asynchronous FIFOs, during the design of the synchronizer, it was assumed that the slower modules will run at least two times slower than the faster modules. This allows us to perceive the slower clock as a normal control signal and treat the chip like it only has a single clock domain. With this approach the current interface from [18] was extended with the clock synchronization modules. The increase in area due to the synchronizer will be included in the area of each candidate.

3.4.5 Constraints

After some preliminary back-end experiments of the six designs were performed, four design points mentioned in the metrics section were collected. The designs were first constrained at TradeOff1, due to its highest overall throughput to area ratio. Then, on the biggest floor-plan that the budget allowed, the constraints on maximum frequency was relaxed until the designs could be placed and routed without any error due to routing congestion. The six hashing modules were categorized into three groups based on their achievable frequency. The speed constraint on the faster group (JH and Keccak) was 250 MHz, the medium group (Groestl and SHA256) was 200 MHz, and the slower group (Blake and Skein) was 125 MHz. With these timing constraints, the chip was finished in area of 5 square millimeters with a core dimension 1.656 mm by 1.6565 mm.

3.4.6 Design Environment

The Synopsys suite was used for front end stage ASIC design and used Cadence Virtuoso for the backend. Synopsys Design Compiler was used to map HDL codes into IBM 130nm primitive gates. The typical conditions were used to characterize the stand-cells. The 130nm technology has 8 metal layers, the 5 lower metal layers were used for routing, while the higher metal layers were used for the power network.

The Synopsys IC Compiler was used for place and route. The overall core area utilization ratio was 73%. The utilization area is computed by dividing the total area of active components, such as standard-cells, by the area of the chip core. Initially, a die with smaller area was used, however, it became quite clear that the designs at trade-off1 wouldn't fit, so the area was extended and the speed constraints were relaxed until they fit properly.

The IR drop analysis was also performed by using the Power Network Analyzer within IC Compiler. The Switching Activity Interchangeable Format file and the reference voltage were provided to the analyzer and verified that the greatest voltage drop is less than ten percent of the supply voltage.

Timing and power results were obtained during the post-synthesis stage as well as the post-layout stage. First, IC compiler was used to extract circuit parasitics and generate the signal delay file. Second, the Synopsys VCS was used for post-layout simulation. VCS records the switching activity into a file, from which, PrimeTime was used to estimate timing and power information.

3.5 FPGA Implementation

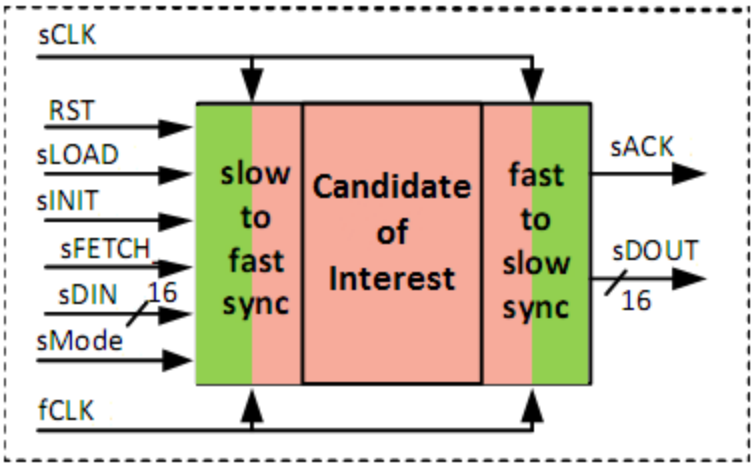

Figure 3.10: An overview of the cryptographic FPGA during prototyping

The FPGA implementation is a break-down of the ASIC implementation. Since not all six candidates need to be on one FPGA device, the system control module and clock management module were removed and each candidate was implemented separately. Figure 3.10 shows the overview of the cryptographic FPGA during the prototyping process. The figure shows that candidates retain their clock synchronization modules during the evaluation for easier comparison across devices.

3.6 Conclusion

This chapter provided detailed information regarding the hardware evaluation project. In the beginning, the methodology of the experiments, including the interface, the metrics and the technology nodes was discussed. The discussion of the CAD flow follows. The description of the process of ASIC and FPGA implementations and the decisions made during the design process were given. The reasons behind the four design points picked for the ASIC implementation were given. After that an overview of the project and the framework of the experiment were described. An overview of the implementation on FPGA as well as on ASIC was provided.

Chapter 4

Hardware Evaluation Results

In this chapter, the results obtained from the ASIC and FPGA experiments will be presented. The results collected from the ASIC device are obtained from post-layout simulation. The results collected from FPGA are obtained from Xflow and ATHENA.

4.1 ASIC Results

Table 4.1: ASIC characterization of the SHA-3 ASIC chip

	Block Size [bits]	Core Latency [cycles]	Area [kGEs]	Max Freq. [MHz]	Throughput [Gbps]	Throughput-to-Area [kbps/GE]	Power [mW]	Energy [mJ/Gbits]
Blake-256	512	30	34.15	120.77	2.06	60.36	17.83	41.78
Groestl-256	512	11	124.34	189.04	8.80	70.76	101.38	87.12
JH-256	512	42	49.29	237.53	2.90	58.75	12.5	41.02
Keccak-256	1024	24	42.49	240.96	10.28	241.97	17.83	16.71
Skein512-256	512	21	66.36	115.07	2.81	42.28	41.75	68.5
SHA-256	512	68	21.67	193.42	1.46	67.21	5.95	31.62

The ASIC results are presented in Table 4.1 the throughput vs. area graph is presented on Figure 4.1, and the throughput-to-area ratio graph is presented Figure 4.2. On the throughput vs. area plot, candidates towards the upper left corner are favorable because they are small and fast, while the candidates towards the lower right corner are unfavorable because they are large and slow.

The candidate Keccak scored the highest in terms of throughput-to-area ratio. In fact it was three times higher than the second highest candidate, Groestl. Keccak is small in size, high in maximum achievable frequency and acceptable in latency. In addition, Keccak has twice the block size as other candidates, which contributes significantly towards its maximum achievable throughput. Together, this allows Keccak to outperform other candidates by a wide margin in ASIC implementations.

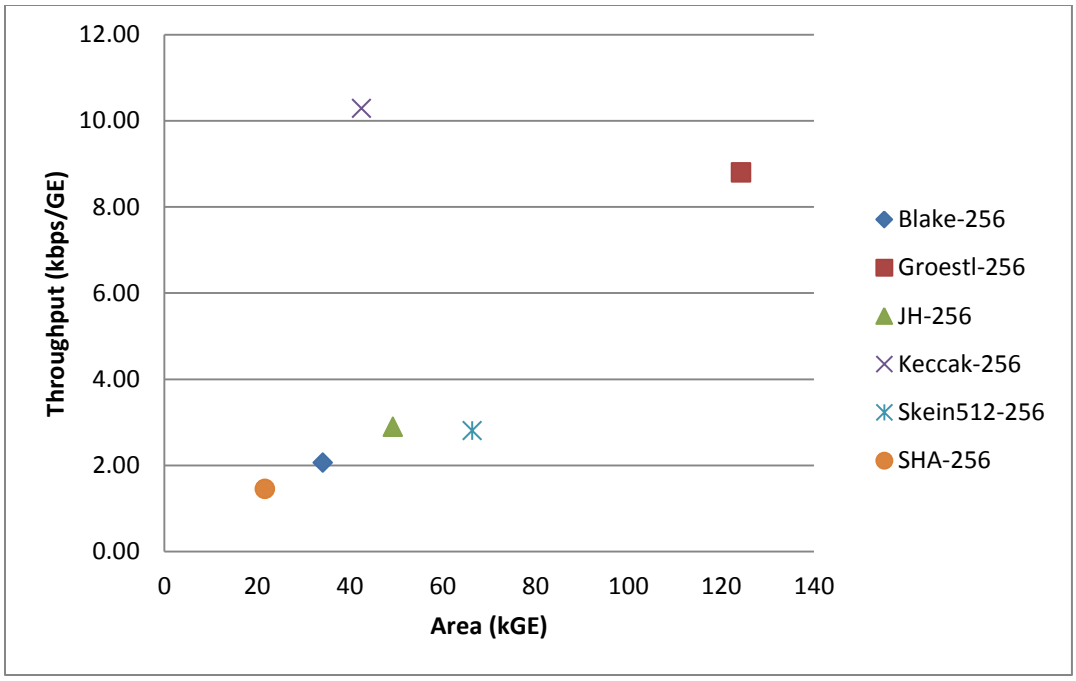

Figure 4.1: Throughput vs. area plot for the SHA-3 ASIC implementation

Candidate Groestl came in second. Groestl is the biggest candidate of all the SHA-3 round three candidates; it's almost twice as big as the next biggest candidate, Skein. However, Groestl has the shortest latency comparing to other candidates, almost half of as much as the next smallest candidate. This makes up for Groestl's large size and mediocre speed and allows it to excel over the three other candidates in terms of throughput-to-area ratio.

Candidate Skein performed less favorably in ASIC implementations. It performed very poorly both in terms of area and achievable speed. It can be seen that Skein's maximum throughput is lower than JH while being moderately larger in area. Despite of Skein's relatively low latency, it still made the lowest mark with throughput-to-area ratio.

Finally, in terms of throughput vs. area, candidate Keccak and Blake will be on the Pareto curve for no candidate has a higher throughput than Keccan and no candidate is smaller in size than Blake.

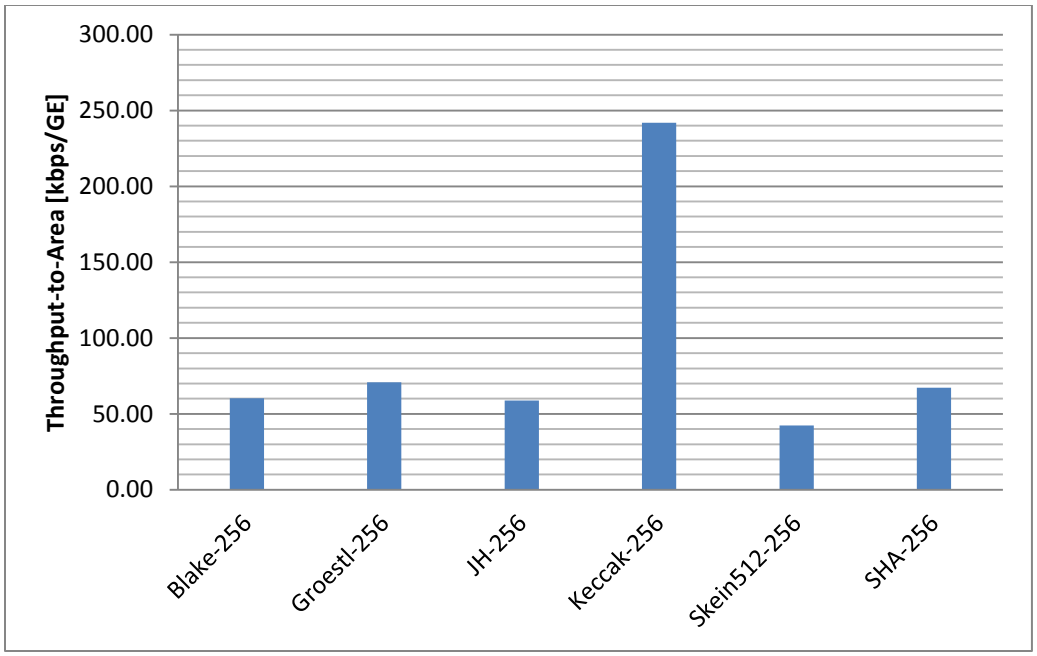

Figure 4.2: Throughput-to-area ratio for the SHA-3 ASIC implementation

Figure 4.3 and 4.4 present the throughput verses power and throughput verses energy. On these graphs, candidates towards the upper left corner are favored for they are more power and energy efficient, while the candidates towards the lower right corner are not favored for they consume large amount of power or energy while processing little data.

Candidate Keccak is both power and energy efficient for its high throughput. Groestl on the other hand, although being fast, consumes a lot of power and energy, which may be ascribed to its large size.

Other designs with lower throughput are consume very little power and moderate amount of energy.

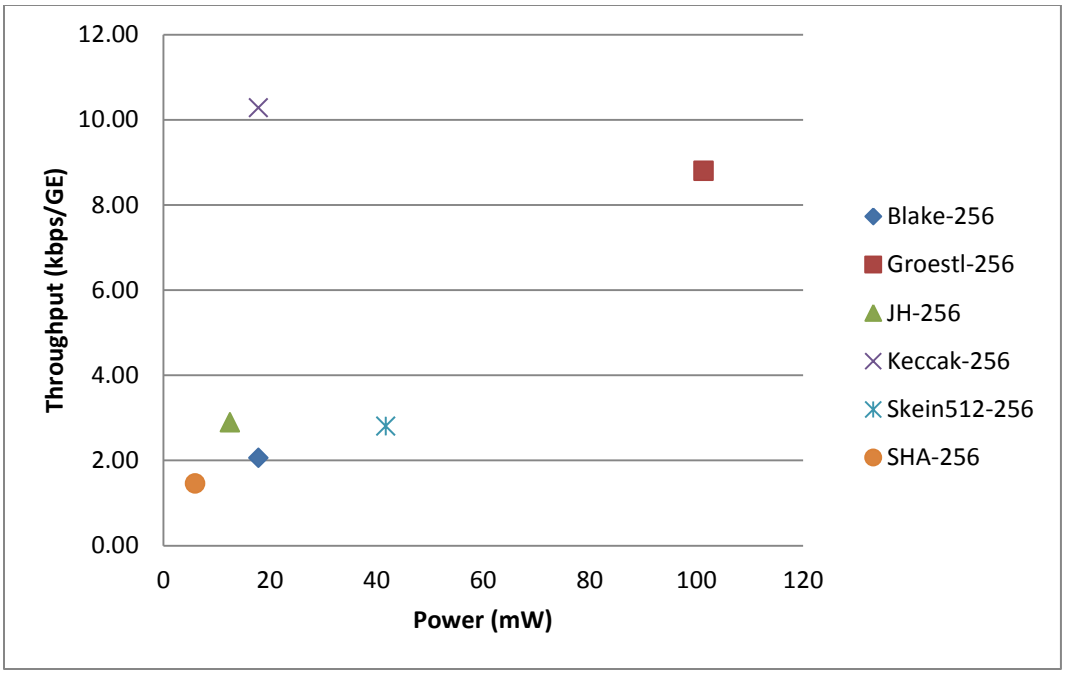

Figure 4.3: Throughput vs. power plot for the SHA-3 ASIC implementation

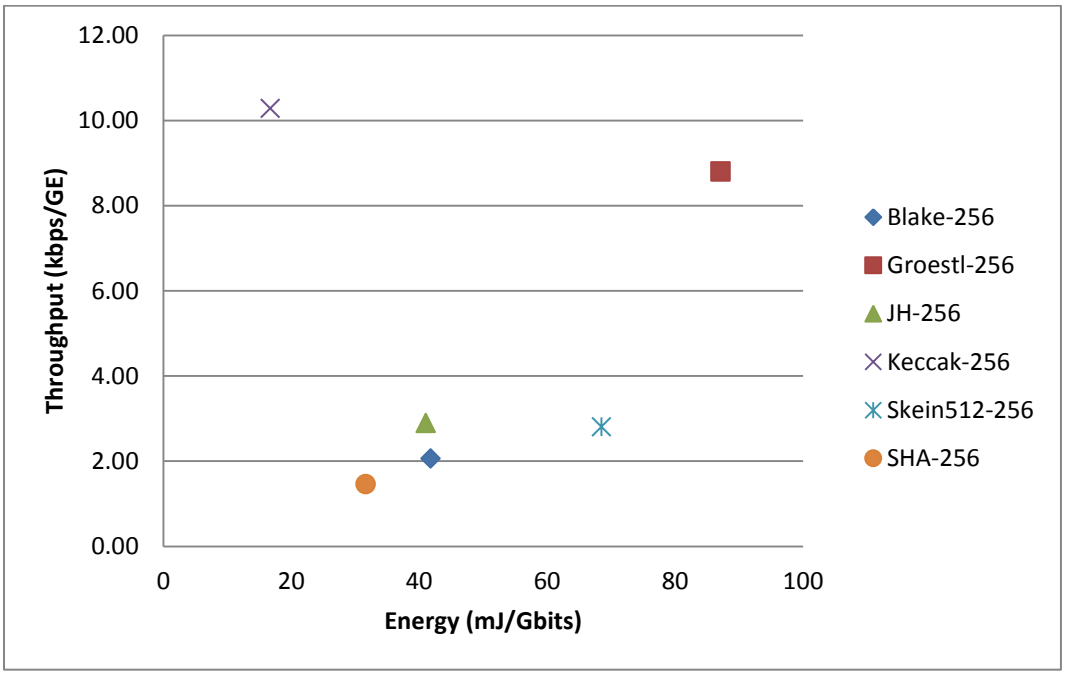

Figure 4.4: Throughput vs. energy plot for the SHA-3 ASIC implementation

4.2 FPGA Results

The FPGA results are presented in this section. The Xflow results are collected from the Xflow reports. The ATHENA flow tries twenty different implementations for each design. The implementations with the highest throughput to area ratio were chosen to represent that design.

4.2.1 Xflow

Table 4.2: FPGA characterization of individual candidates using Xflow

		Block Size [bits]	Core Latency [cycles]	Area [Slices]	Max Freq. [MHz]	Throughput [Gbps]	Throughput-to-Area [kbps/Slice]	Power [mW]	Energy [mJ/Gbits]
Spartan 3	Blake-256	512	30	3949	33.82	0.58	0.15	235.84	690.92
	Groestl-256	512	11	10212	31.12	1.45	0.14	1889.26	2029.48
	JH-256	512	42	5081	74.94	0.91	0.18	131.57	539.66
	Keccak-256	1024	24	4087	58.01	2.63	0.64	186.37	205.55
	Skein512-256	512	21	3502	36.39	0.89	0.25	280.47	575.17
	SHA-256	512	68	2027	79.74	0.60	0.30	44.15	293.19
Virtex 4	Blake-256	512	30	4060	57.67	0.98	0.24	398.68	1168.02
	Groestl-256	512	11	10639	70.67	3.29	0.31	2064.18	2217.38
	JH-256	512	42	6316	110.34	1.35	0.21	368.17	1510.08
	Keccak-256	1024	24	4584	122.03	5.53	1.21	384.86	424.48
	Skein512-256	512	21	3739	59.69	1.46	0.39	503.12	1031.80
	SHA-256	512	68	2102	141.60	1.07	0.51	125.67	834.56
Virtex 5	Blake-256	512	30	1934	77.15	1.32	0.68	276.94	811.36
	Groestl-256	512	11	4980	84.34	3.93	0.79	1209.41	1299.17
	JH-256	512	42	2984	144.55	1.76	0.59	318.73	1307.31
	Keccak-256	1024	24	1684	116.66	5.29	3.14	289.74	319.56
	Skein512-256	512	21	1752	55.89	1.36	0.78	471.19	966.30
	SHA-256	512	68	843	164.15	1.24	1.47	95.33	633.03
Virtex 6	Blake-256	512	30	1934	84.60	1.44	0.75	189.39	554.85
	Groestl-256	512	11	4614	116.84	5.44	1.18	799.52	858.86
	JH-256	512	42	2099	157.51	1.92	0.91	132.60	543.88
	Keccak-256	1024	24	1942	113.82	5.16	2.66	244.61	269.79
	Skein512-256	512	21	1618	89.75	2.19	1.35	237.19	486.43
	SHA-256	512	68	507	183.15	1.38	2.72	34.66	230.14

Table 4.2 shows the candidates' performance on FPGAs using Xflow. Since, with Xflow, most of the settings are set at default, not all the candidates' performances are being equally represented. However, some trends are still evident.

Figure 4.5 shows the area of each candidate in different FPGA platforms. Figure 4.6 shows the throughput while Figure 4.7 shows the throughput-to-area ratio.

Across different FPGA platforms, Keccak and reference SHA-2 are consistently the best performers, while Blake and JH are typically poor performers. The reference SHA-2 algorithm is always the top performer because it has the smallest size and highest achievable frequency on every platform. Keccak on the other hand, despite only having mediocre numbers in area, speed and latency, is still one of the top performers across all platforms of FPGA. A merit of Keccak, shown in the results of Xflow, is that while its area, speed or latency results are not the best, none of them are poor either. Take JH for example, the hash candidate with the second highest maximum frequency on all platforms but Virtex 5. Despite that its speed is often on par with SHA-2, its results in area and especially in latency brought down its overall performance. Blake on the other hand is impressively small but its poor clock speed and long latency make it less favorable.

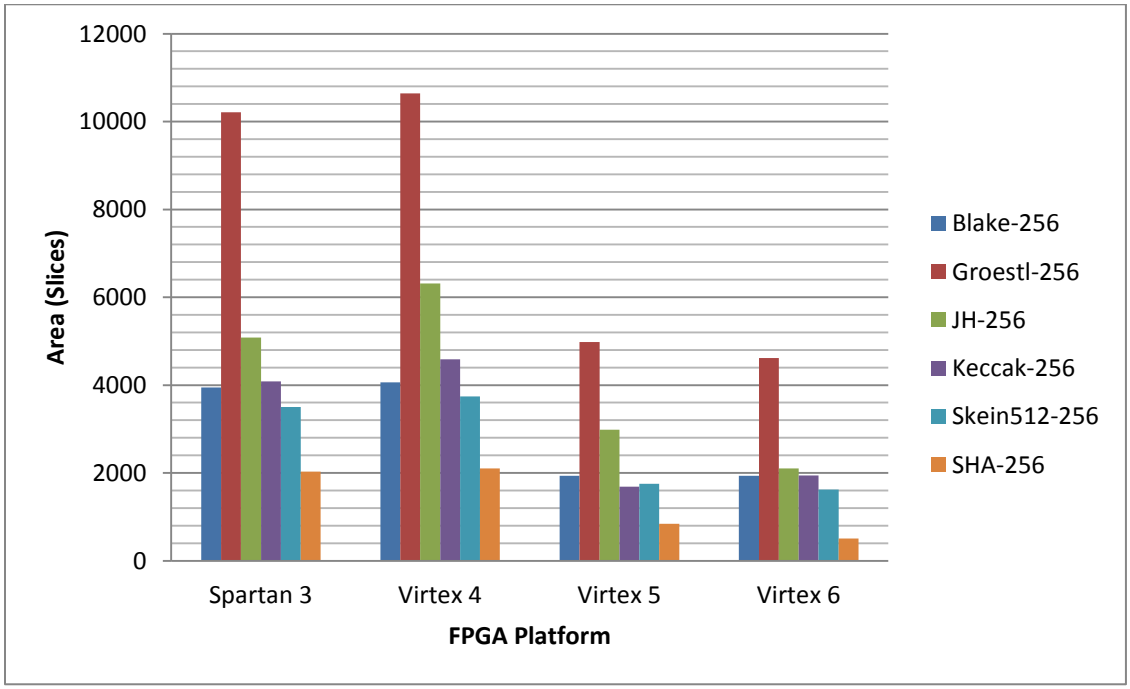

Figure 4.5: Area results for the hash candidates on the FPGAs with Xflow

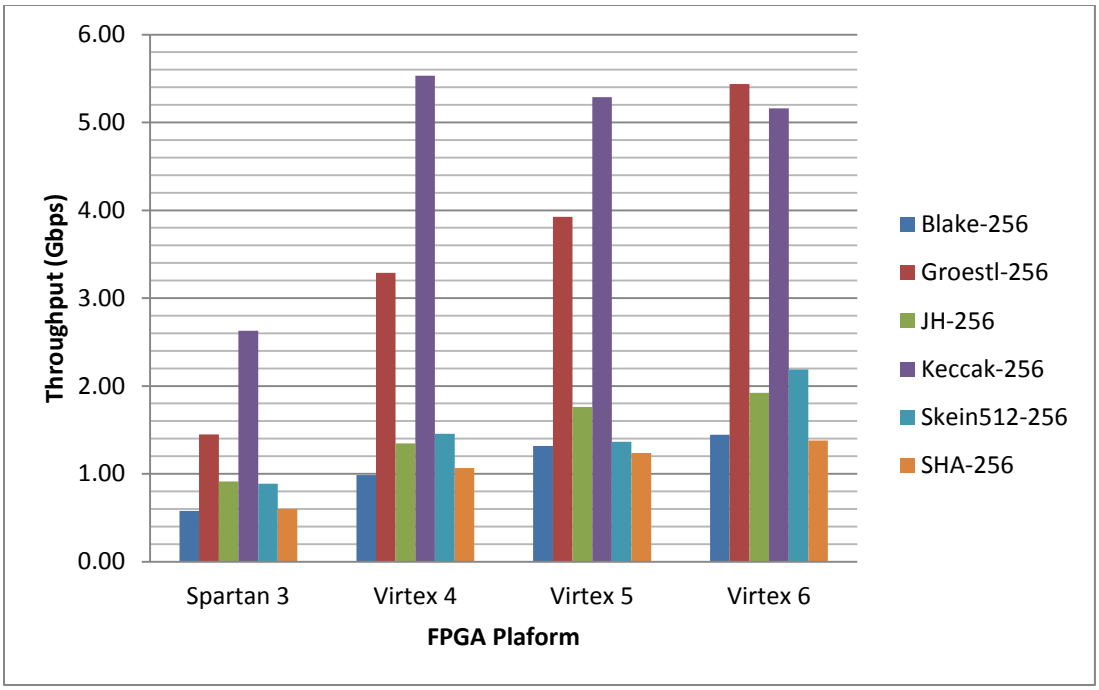

Figure 4.6: Throughput results for the hash candidates on the FPGAs with Xflow

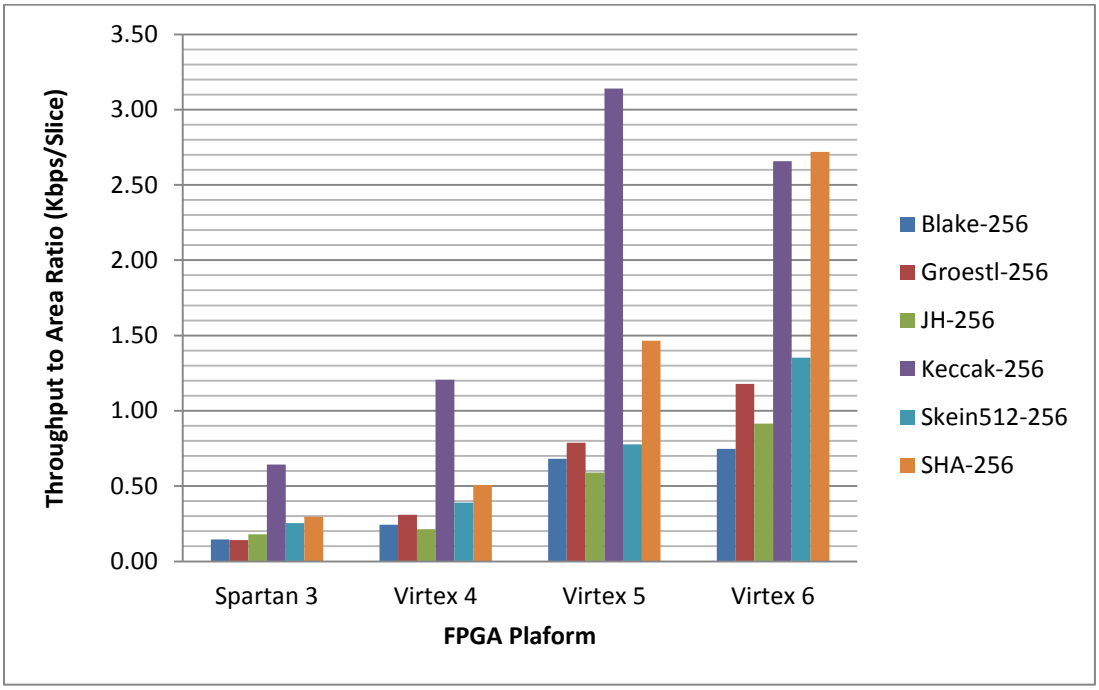

Figure 4.7: Throughput vs. area results for the hash candidates on the FPGAs with Xflow

Figure 4.8 and 4.9 show the power and energy results.

As the case with ASIC, Groestl still has huge power and energy dissipation on FPGA due to its large size, which in turn translates to large static power dissipation. JH and Blake seem to have low power dissipation but relatively high energy dissipation. This may be ascribed to JH and Blake's large latency.

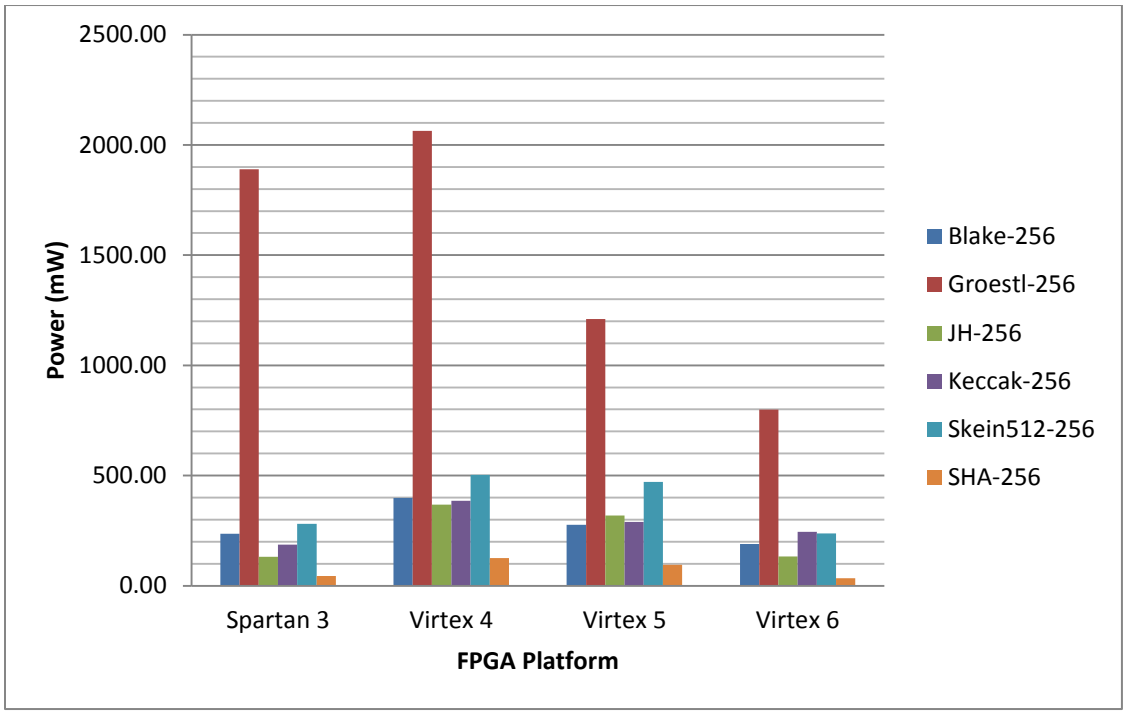

Figure 4.8: Power results for the hash candidates on the FPGAs with Xflow

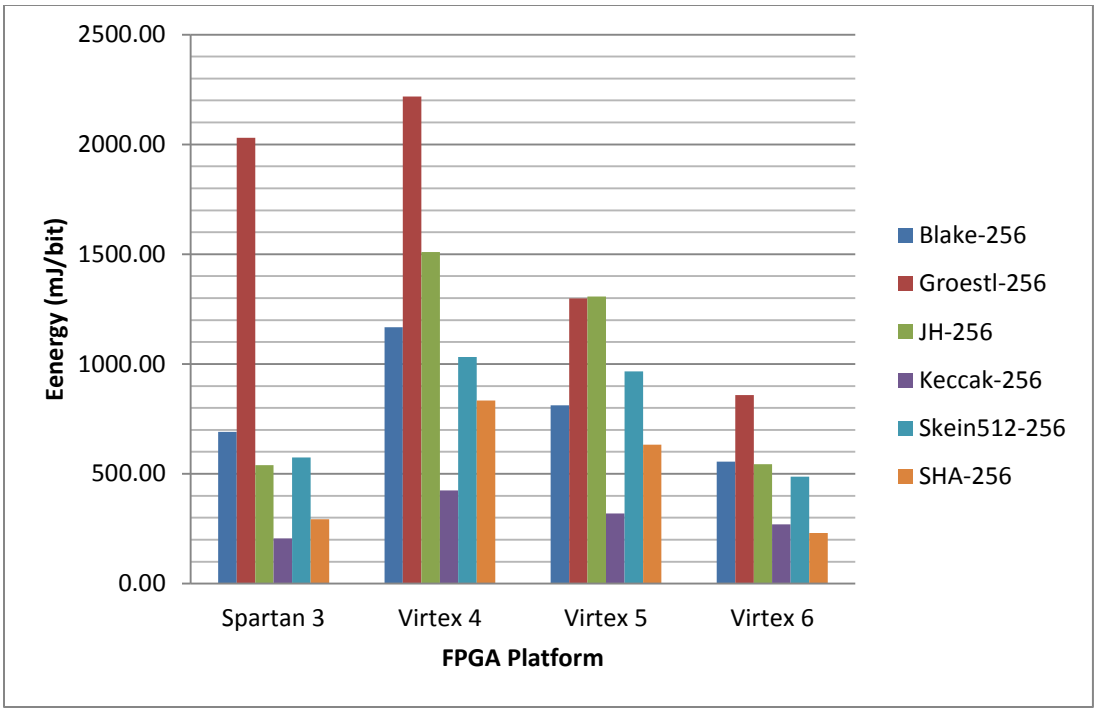

Figure 4.9: Energy results for the hash candidates on the FPGAs with Xflow

4.2.2 ATHENA flow

Table 4.3: FPGA characterization of individual candidates using ATHENA

		Block Size [bits]	Core Latency [cycles]	Area [Slices]	Max Freq. [MHz]	Throughput [Gbps]	Throughput-to-Area [kbps/Slice]	Power [mW]	Energy [mJ/Gbits]
Spartan 3	Blake-256	512	30	4030	37.54	0.64	0.16	237.38	695.46
	Groestl-256	512	11	9858	39.58	1.84	0.19	1624.40	1744.96
	JH-256	512	42	5103	94.34	1.15	0.23	140.49	576.25
	Keccak-256	1024	24	3922	88.88	4.03	1.03	242.79	267.78
	Skein512-256	512	21	3431	35.17	0.86	0.25	296.44	607.94
	SHA-256	512	68	2036	85.07	0.64	0.31	56.14	372.83
Virtex 4	Blake-256	512	30	4063	72.05	1.23	0.30	374.31	1096.62
	Groestl-256	512	11	10877	85.63	3.99	0.37	2193.94	2356.77
	JH-256	512	42	5250	172.41	2.10	0.40	287.06	1177.38
	Keccak-256	1024	24	3922	176.74	8.01	2.04	368.04	405.93
	Skein512-256	512	21	3460	70.01	1.71	0.49	417.99	857.21
	SHA-256	512	68	2126	188.04	1.42	0.67	122.83	815.69
Virtex 5	Blake-256	512	30	1835	110.24	1.88	1.03	300.28	879.73
	Groestl-256	512	11	3798	108.00	5.03	1.32	1205.71	1295.20
	JH-256	512	42	1576	244.38	2.98	1.89	200.03	820.42
	Keccak-256	1024	24	1553	220.46	9.99	6.44	322.90	356.14
	Skein512-256	512	21	1613	104.75	2.55	1.58	363.25	744.95
	SHA-256	512	68	744	251.51	1.89	2.55	87.49	580.99
Virtex 6	Blake-256	512	30	1591	121.18	2.07	1.30	245.66	719.71
	Groestl-256	512	11	3050	102.40	4.77	1.56	962.38	1033.81
	JH-256	512	42	1188	268.89	3.28	2.76	170.45	699.10
	Keccak-256	1024	24	1440	194.14	8.80	6.11	266.87	294.34
	Skein512-256	512	21	1396	113.30	2.76	1.98	311.71	639.25
	SHA-256	512	68	469	309.60	2.33	4.97	59.04	392.09

Table 4.3 shows the candidates' performance on FPGAs using the ATHENA flow. Since the ATHENA flow performs constraint explorations and cost table optimizations, the results by ATHENA provide a better representation of the candidates' performance.

Figure 4.10 shows the area of each candidate in different FPGA platforms. Figure 4.11 shows the throughput while Figure 4.12 shows the throughput-to-area ratio. Figure 4.13 and 4.14 show the power and energy results.

Across different platforms, the ranking of candidates are the same. Keccak and reference SHA-2 are the best while Groestl and Blake are the worst. It can be seen that across all platforms, both Keccak and

SHA-2 are strong performers by having very competitive numbers in both size and speed. Like the case with Xflow, Skein and JH both have impressive results in one metric, while poor results in others. Skein is a small design with low latency. However, its speed is often the worst of all SHA-3 candidates. JH on the other hand is a very fast candidate, but its large size and long latency makes it less competitive. Groestl has high throughput, however, its enormous size and less than average maximum clock speed makes it one of the worst performers on FPGA. Blake is lacking on both in terms of area and throughput, thus making it the worst performer of the experiment.

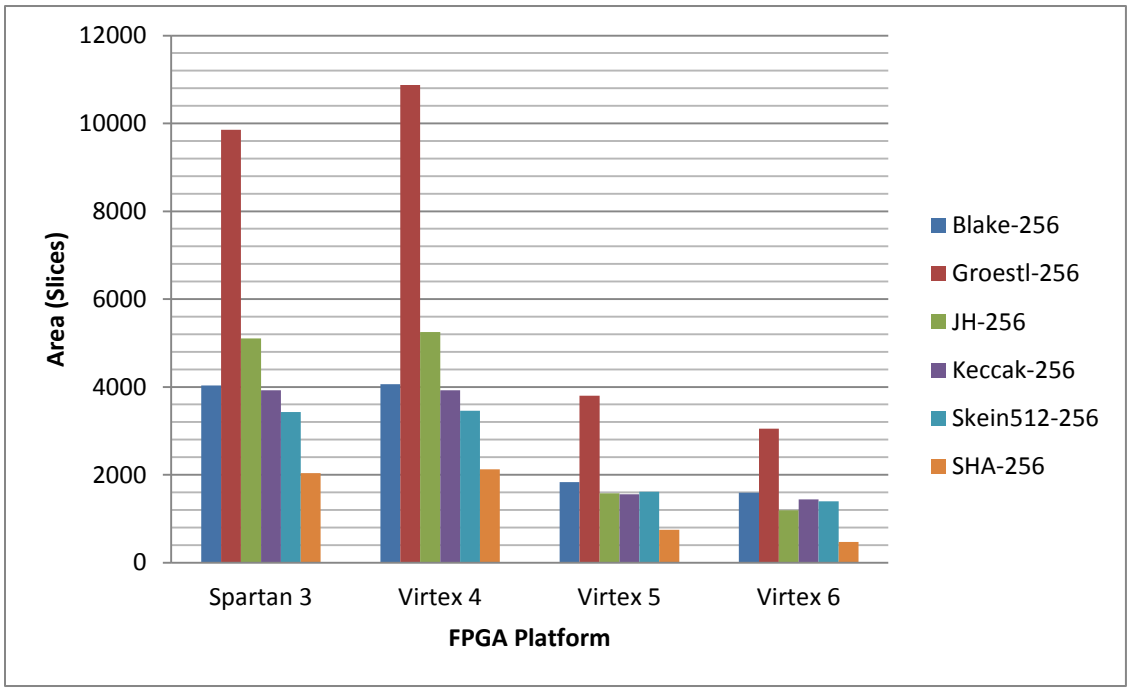

Figure 4.10: Area results for the hash candidates on the FPGAs with ATHENA flow

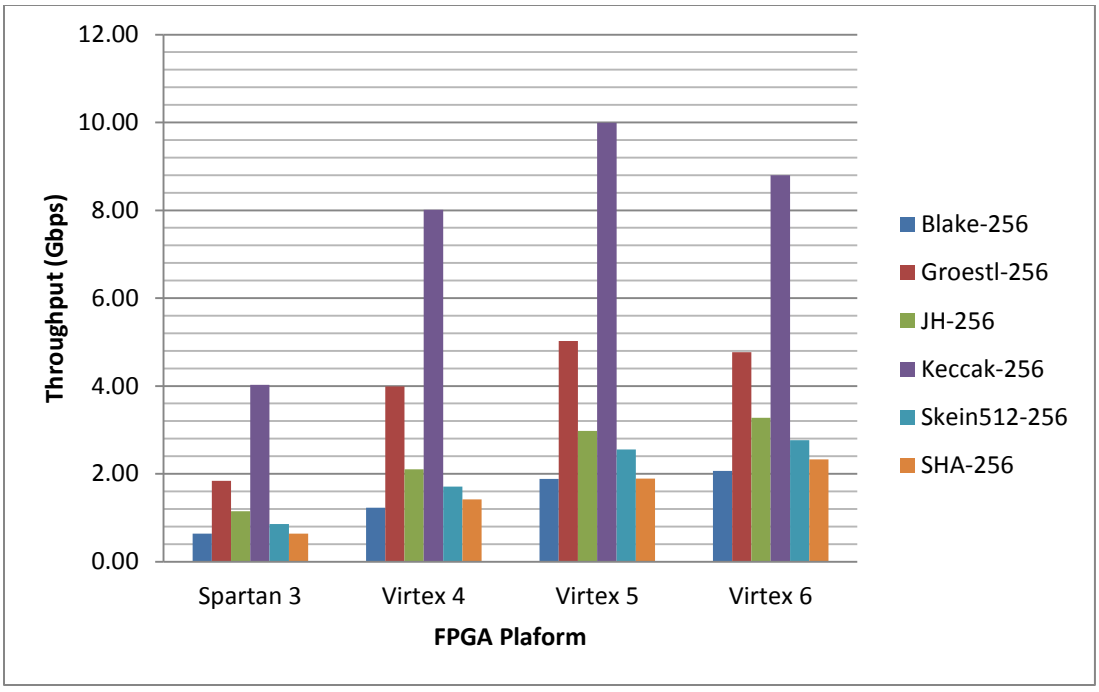

Figure 4.11: Throughput results for the hash candidates on the FPGAs with ATHENA flow

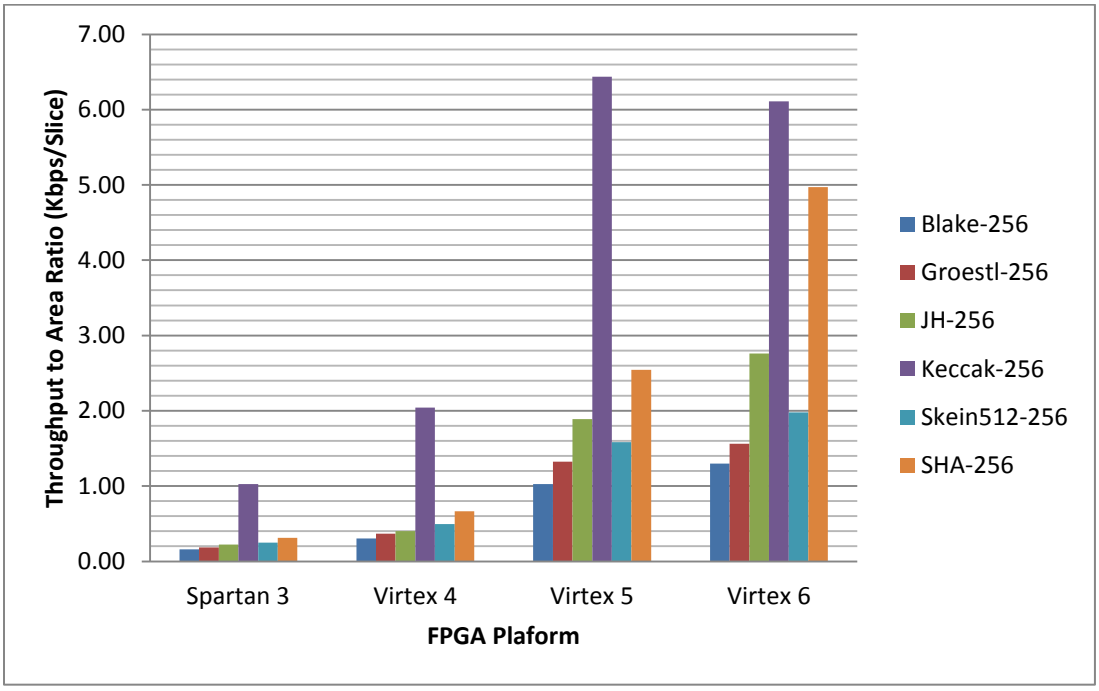

Figure 4.12: Throughput vs. area results for the hash candidates on the FPGAs with ATHENA flow

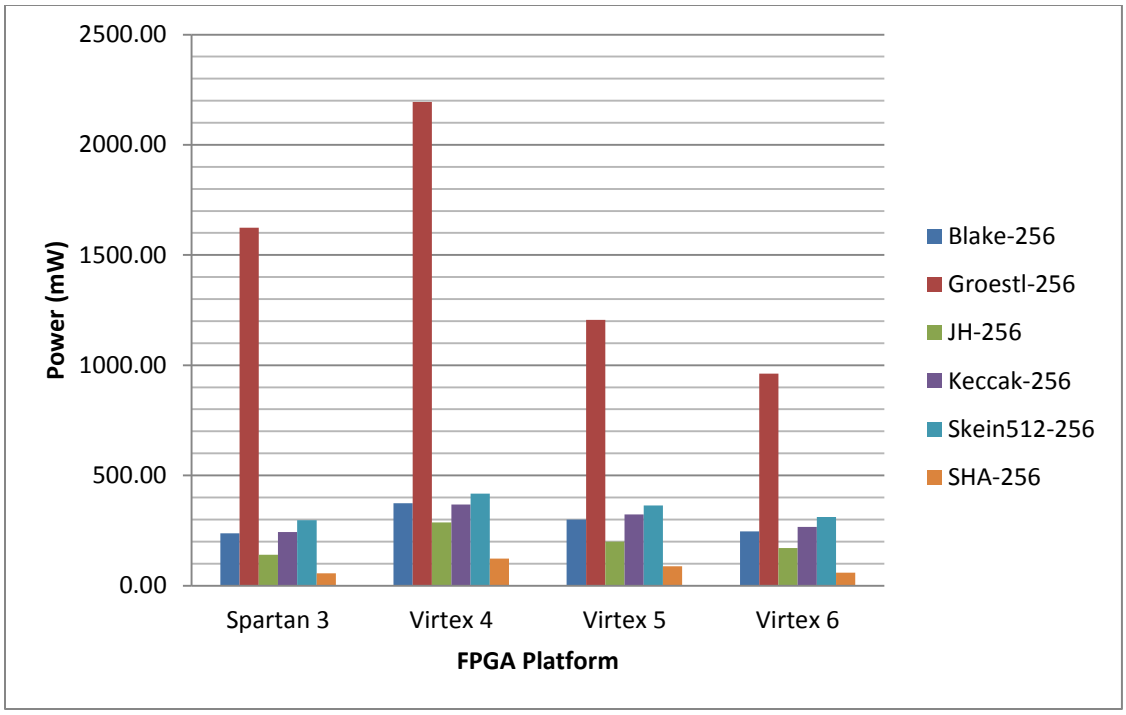

Figure 4.13: Power results for the hash candidates on the FPGAs with ATHENA flow

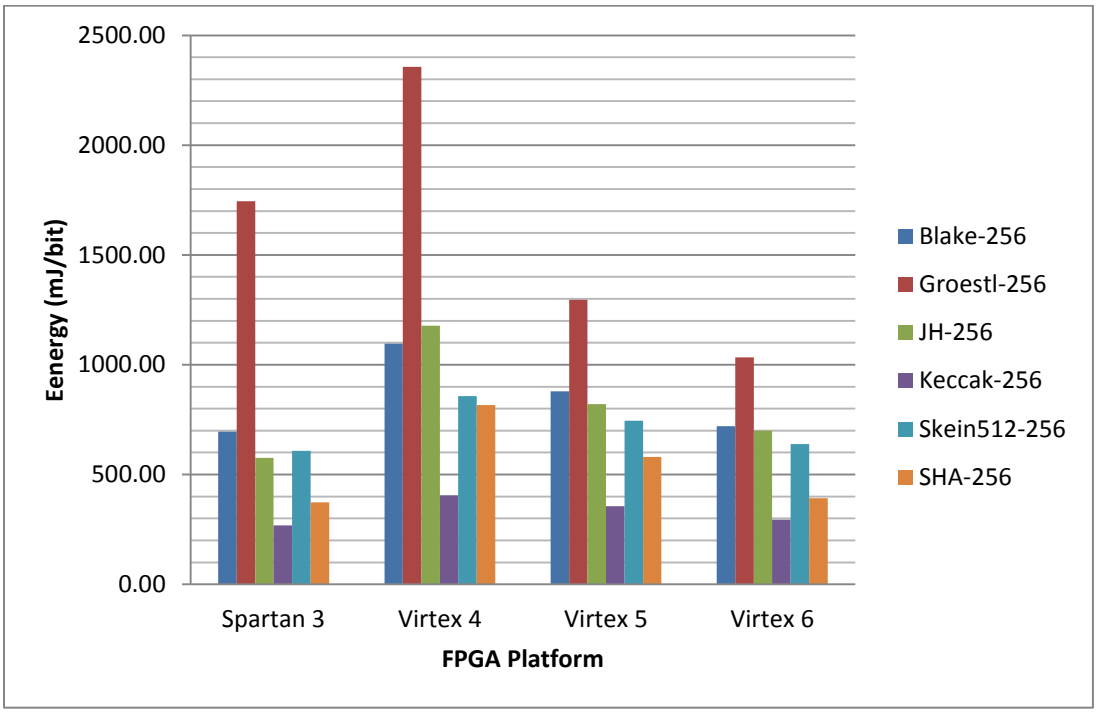

Figure 4.14: Energy results for the hash candidates on the FPGAs with ATHENA flow

4.2.3 Comparison between Xflow and ATHENA flow

Table 4.4: Comparison between FPGA ATHENA flow and Xflow on Virtex 5

ATHENA

	Block Size [bits]	Core Latency [cycles]	Area [Slices]	Max Freq. [MHz]	Throughput [Gbps]	Throughput-to-Area [kbps/Slice]	Power [mW]	Energy [mJ/Gbits]
Blake-256	512	30	1835	110.24	1.88	1.03	300.28	879.73
Groestl-256	512	11	3798	108.00	5.03	1.32	1205.71	1295.20
JH-256	512	42	1576	244.38	2.98	1.89	200.03	820.42
Keccak-256	1024	24	1553	220.46	9.99	6.44	322.90	356.14
Skein512-256	512	21	1613	104.75	2.55	1.58	363.25	744.95
SHA-256	512	68	744	251.51	1.89	2.55	87.49	580.99

Xflow

	Block Size [bits]	Core Latency [cycles]	Area [Slices]	Max Freq. [MHz]	Throughput [Gbps]	Throughput-to-Area [kbps/Slice]	Power [mW]	Energy [mJ/Gbits]
Blake-256	512	30	1934	77.15	1.32	0.68	276.94	811.36
Groestl-256	512	11	4980	84.34	3.93	0.79	1209.41	1299.17
JH-256	512	42	2984	144.55	1.76	0.59	318.73	1307.31
Keccak-256	1024	24	1684	116.66	5.29	3.14	289.74	319.56
Skein512-256	512	21	1752	55.89	1.36	0.78	471.19	966.30
SHA-256	512	68	843	164.15	1.24	1.47	95.33	633.03

In this section, the comparison of the results between the two FPGA flows will be shown. The results are shown in Table 4.4. The implementation FPGA is Virtex 5. Figure 4.15 shows the throughput to area ratio results normalized to SHA-2 from both flows. Figure 4.16 and 4.17 show the area and throughput results on the Virtex 5 from Xflow and ATHENA flow.

From Figure 4.15, it appears that the overall ranking of the candidates remain largely the same across the two flows. Two candidates that show the greatest displacement are JH and Keccak. The reason for this can be seen in Figure 4.16 and 4.17. When moving from Xflow to ATHENA flow, the area of JH halved while the throughput of both JH and Keccak almost doubled. This explains why the performance of JH and Keccak differ across the two flows. In addition, candidates Blake and Groestl appear to have changed places, however this is because their performance numbers are very close on the Xflow, and ATHENA didn't yield as much improvement as with JH and Keccak. Thus the performance results for these candidates remained almost the same but traded places with one another.

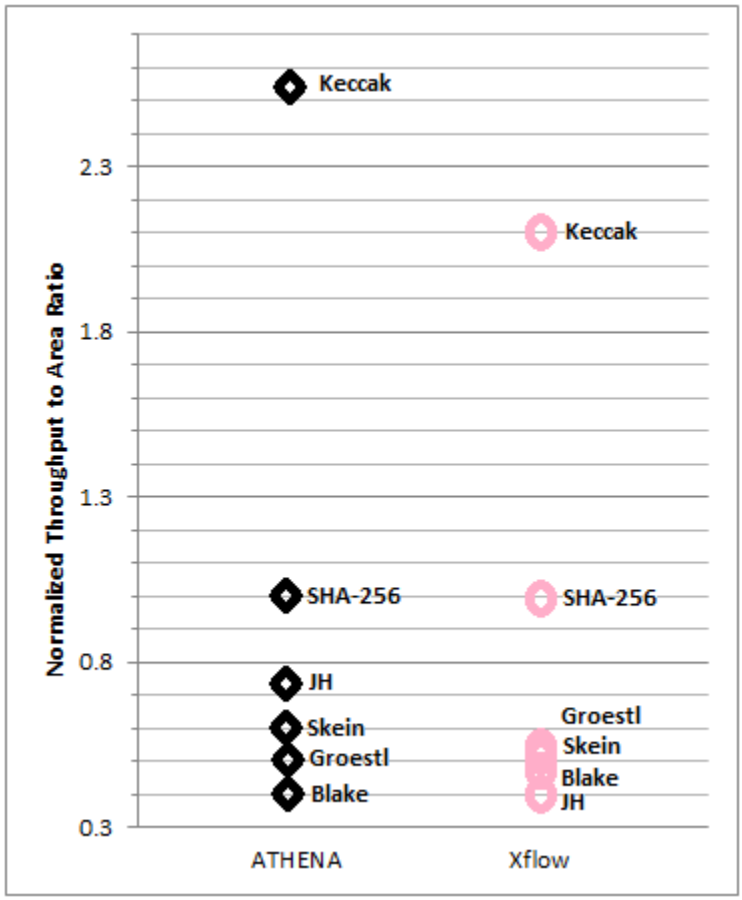

Figure 4.15: Comparison of the throughput to area ratio results normalized to SHA-256 for the hash candidates on Virtex 5 between ATHENA and Xflow

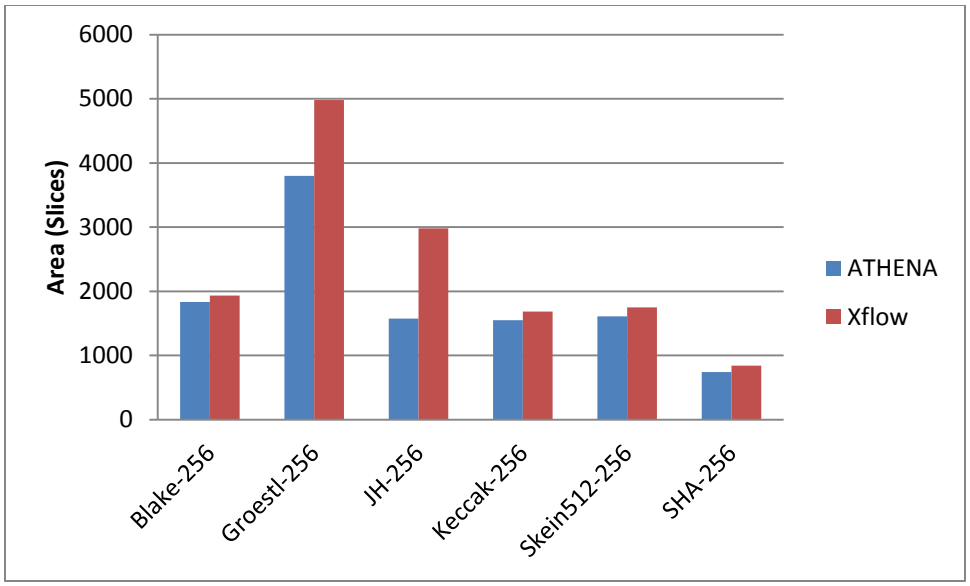

Figure 4.16: Comparison of the area results for the hash candidates on Virtex 5 between ATHENA and Xflow

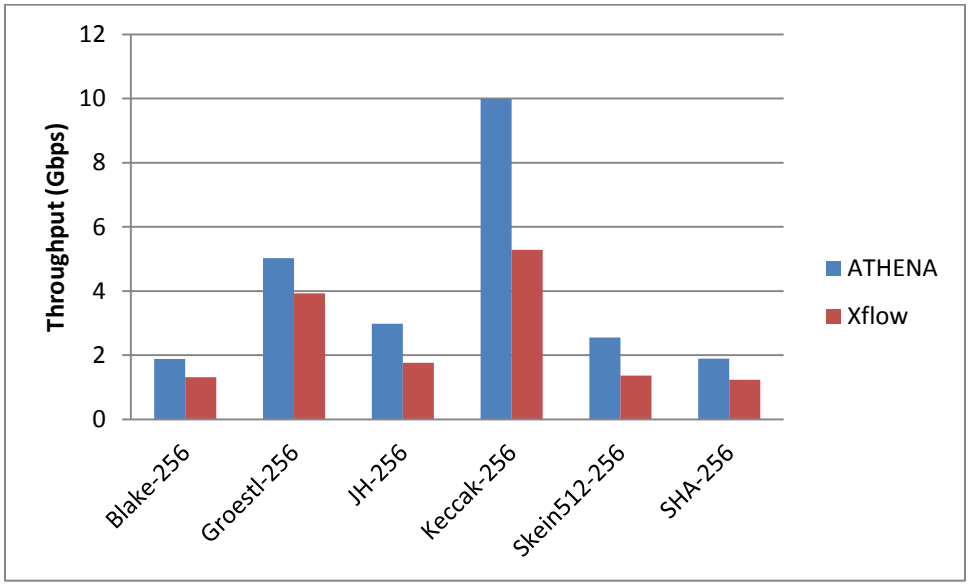

Figure 4.17: Comparison of the throughput results for the hash candidates on Virtex 5 between ATHENA and Xflow

Figure 4.18 and 4.19 show the power and energy results from both flows. It appears that the ATHENA's improvement on power and energy dissipation of the candidates isn't as high as performance improvement. In fact, the power consumption for Blake actually deteriorated. Candidate JH has the greatest improvement from Xflow to ATHENA flow, due to the large improvement over its area, as

mentioned before. The reduction in size for JH helps decrease the static power dissipation, which in turn improved its energy dissipation results.

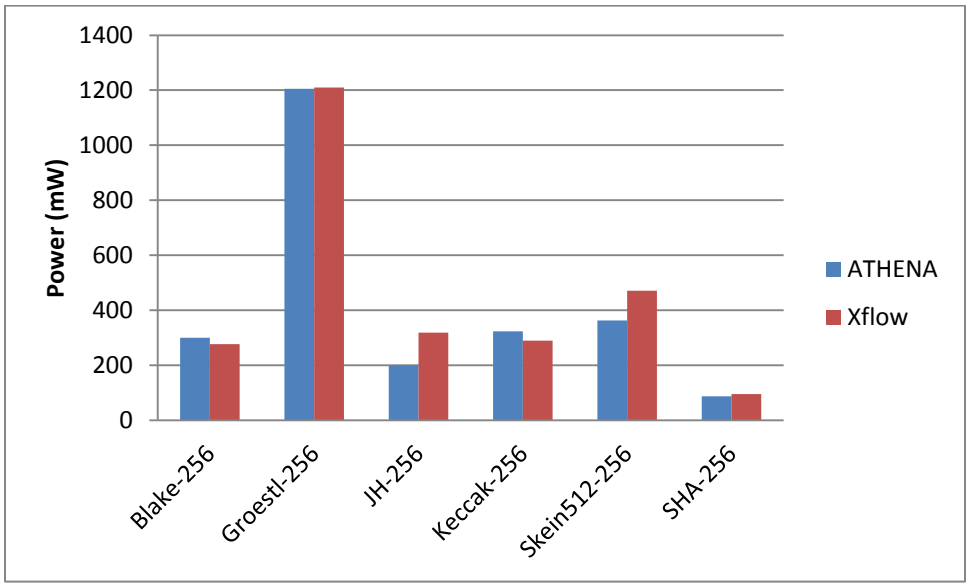

Figure 4.18: Comparison of the power results for the hash candidates on Virtex 5 between ATHENA and Xflow

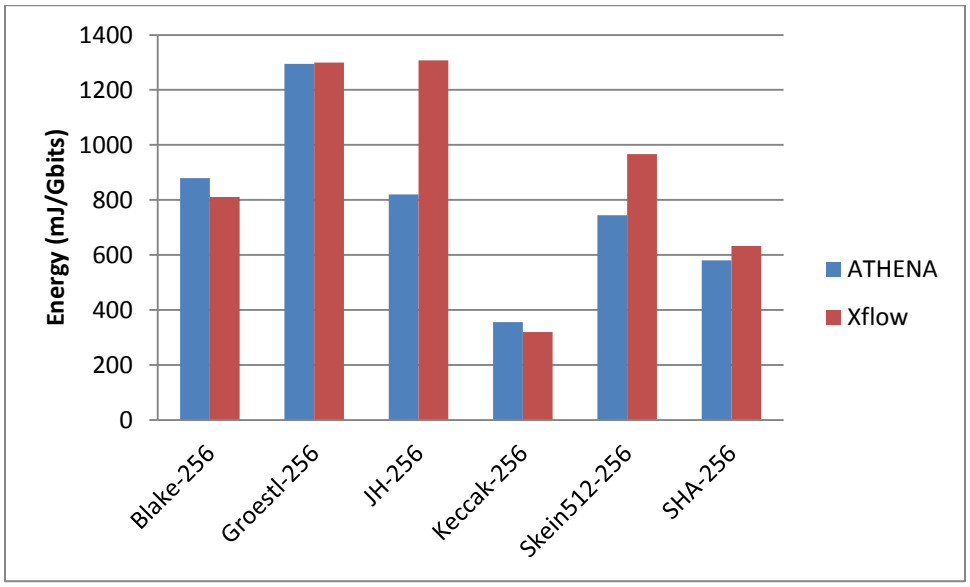

Figure 4.19: Comparison of the energy results for the hash candidates on Virtex 5 between ATHENA and Xflow

In summary, it is shown that from Xflow to ATHENA flow, the area and throughput results of the candidates improve by various amounts. Therefore, the rankings may change accordingly. In addition, ATHENA seems to have relatively minor improvements on power and energy reduction. This comparison

shows that using default options and constraints on Xflow, candidates Blake, Groestl and Skein are adequately represented while JH and Keccak are underrepresented.

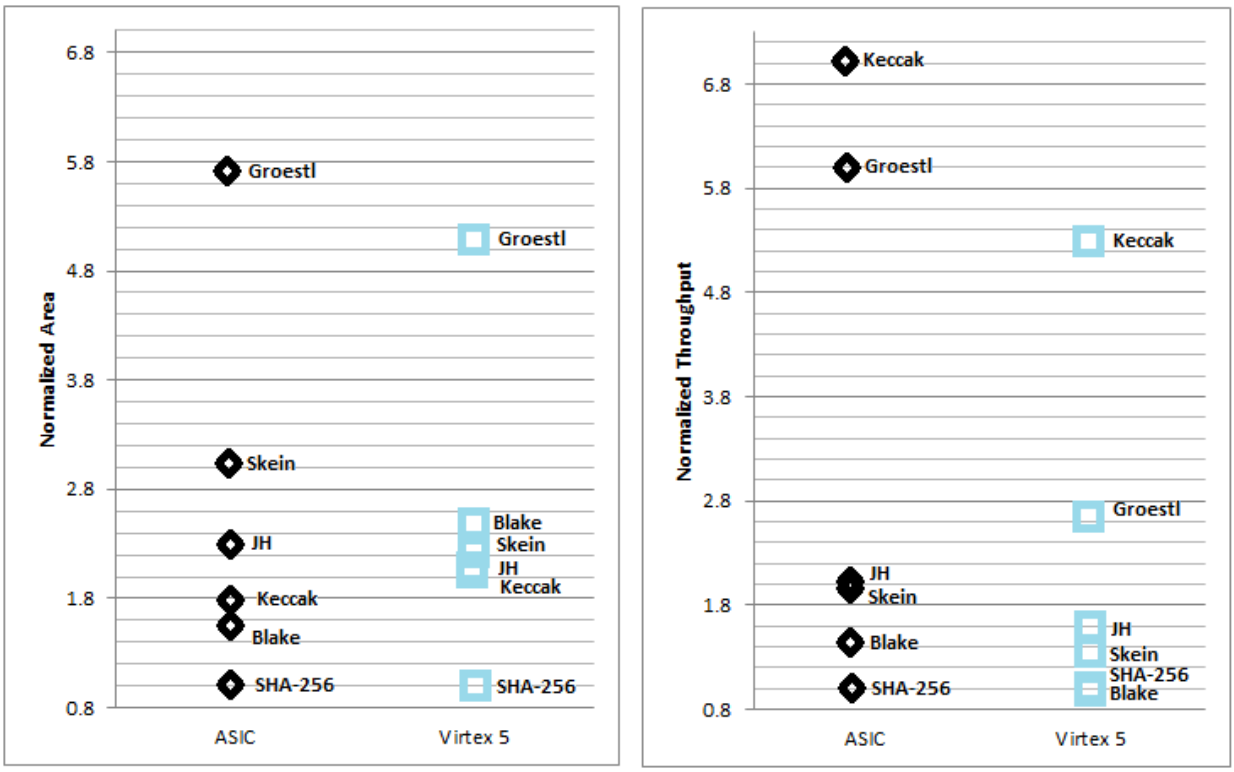

Figure 4.20: Comparison of the area and throughput results normalized to SHA-2 for the hash candidates on Virtex 5 between ATHENA and Xflow

4.2.4 Comparison ASIC and FPGA ATHENA flow

Table 4.5: Comparison between FPGA ATHENA flow and ASIC, for ATHENA flow Virtex 5 was used

ATHENA

	Block Size [bits]	Core Latency [cycles]	Area [Slices]	Max Freq. [MHz]	Throughput [Gbps]	Throughput-to-Area [kbps/Slice]	Power [mW]	Energy [mJ/Gbits]
Blake-256	512	30	1835	110.24	1.88	1.03	300.28	879.73
Groestl-256	512	11	3798	108.00	5.03	1.32	1205.71	1295.20
JH-256	512	42	1576	244.38	2.98	1.89	200.03	820.42
Keccak-256	1024	24	1553	220.46	9.99	6.44	322.90	356.14
Skein512-256	512	21	1613	104.75	2.55	1.58	363.25	744.95
SHA-256	512	68	744	251.51	1.89	2.55	87.49	580.99

ASIC

	Block Size [bits]	Core Latency [cycles]	Area [kGEs]	Max Freq. [MHz]	Throughput [Gbps]	Throughput-to-Area [kbps/GE]	Power [mW]	Energy [mJ/Gbits]
Blake-256	512	30	34.15	120.77	2.06	60.36	17.83	41.78
Groestl-256	512	11	124.34	189.04	8.80	70.76	101.38	87.12
JH-256	512	42	49.29	237.53	2.90	58.75	12.50	41.02
Keccak-256	1024	24	42.49	240.96	10.28	241.97	17.83	16.71
Skein512-256	512	21	66.36	115.07	2.81	42.28	41.75	68.50
SHA-256	512	68	21.67	193.42	1.46	67.21	5.95	31.62

In this section, results between the ASIC flow and the FPGA ATHENA flow will be compared. For the FPGA flow, Virtex 5 was picked as the FPGA platform for comparison. The results are shown in Table 4.5. Figure 4.20 shows the area and throughput results on ASIC and Virtex 5 FPGA with ATHENA flow normalized to SHA-2.

Across the hardware devices of FPGA and ASIC, moderate changes appear in the rankings. In ASIC, Keccak is clearly the fastest candidate in terms of maximum clock speed, however on Virtex 5, the maximum achievable frequency of Keccak is surpassed by SHA-2, even though on ASIC, SHA-2 was placed in the medium group while Keccak was in the fast group. The candidate with the greatest change in ranking across FPGA and ASIC is Groestl. In ASIC, Groestl's performance comes in second, on FPGA however, Groestl is the second worst in terms of performance. The reason for that is because moving from ASIC to FPGA, Groestl has retained its large size but lost its impressive speed. In the next chapter, another implementation of Groestl is attempted to improve the result on the FPGAs. The results for Blake

changed significantly across devices. On ASIC, Blake is the smallest SHA-3 candidate, on the FPGA however, Blake is the second biggest candidate next to Groestl. In addition, the relative maximum frequency for Blake has also deteriorated moving from ASIC to FPGA. The cause of this is unknown. The results of other candidates remain largely in their place across both platforms.

In summary, the results between the ASIC implementations and the implementations on Virtex5 with ATHENA flow were briefly compared. It appears that Keccak is the most favored candidate across the devices, and results for Blake and Groestl have deteriorated in moving from ASIC to FPGA.

4.3 Conclusion

In this chapter, the results in both ASIC and FPGA were shown. It can be seen that Keccak has performed best in both FPGA and ASIC due to its small size and high throughput. Groestl on the other hand, performed well in ASIC with its high throughput but not in FPGA due to its large size. JH and Skein both performed moderately well in FPGA and ASIC while Blake performed poorly on both.

Chapter 5

Additional Studies of Hash Candidates

From the previous chapter, the SHA-3 candidates implemented by this group on the hardware devices were evaluated. In this chapter, additional experiments will be described and their results will be shown. Based on these results, some new avenues are suggested to explore and make the hardware evaluation more complete.

5.1 Optimization of Groestl

In this section, an optimization of hash candidate Groestl will be shown. As seen in the previous chapter, Groestl performs well in ASIC; however, in an FPGA Groestl is undesirable because of its large size and slow speed. This experiment attempts to improve the result of Groestl on the FPGA.

5.1.1 Round Function of Groestl

An overview was given of the structure of Groestl in the previous chapter. This section presents a more detailed look into Groestl's round function.

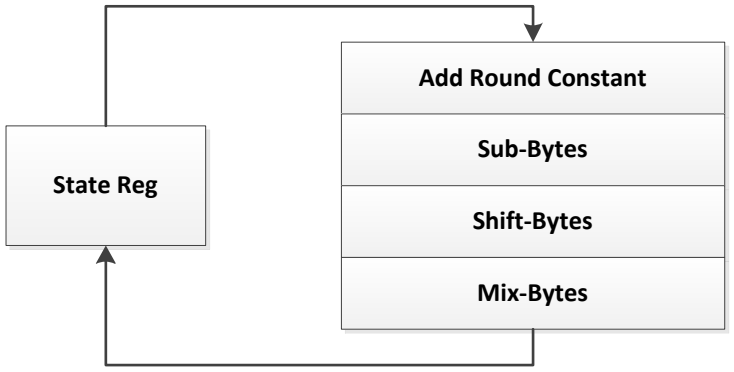

Figure 5.1: An overview of the round function of Groestl [22]

Figure 5.1, presents the round function of Groestl. The gridded squares are the internal state registers. Four operations take place in the round function: AddRoundConstant, SubBytes, ShiftBytes, and MixBytes.

In AddRoundConstant, the states are being XORed with round-dependent constants. In SubBytes, the states go through a non-linear transformation. In ShiftBytes, data are shifted cyclically to the left in a row by different numbers of places, while the MixBytes operations multiply the states by a constant matrix. The optimized operation is the SubByte operation.

5.1.2 Optimization of SubByte

Table 5.1: The S-Box used by Groestl [22]

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Groestl uses an 8-bit S-Box shown on Table 5.1. In software, the SubByte operation is carried out by an implementation of an S-Box. In the original HDL source, the S-Box is implemented in sub-field operation. The detail of this operation is beyond the scope of this thesis. In summary, this operation allows us to directly compute the S-Box function. The logical resource required for this operation is presented in Table 5.2 below. The data is extracted from a Xilinx synthesis report.

Table 5.2: Resources inferred by Xilinx synthesis tool on Groestl's GF based S-Box

Logic Resources	Num
1-bit xor2	48
1-bit xor3	14
1-bit xor4	4
1-bit xor5	2
9-bit xor2	1

In this optimization, the GF-based S-Box was replaced by a ROM-based S-Box using eight 256-bit ROMs. After re-running the ATHENA flow, the results are presented on Table 5.3.

5.1.3 Results and Conclusion

Table 5.3: Implementation results of Groestl with LUT-based S-Box

	Area (LUT based) [Slices]	Max Freq. (LUT based) [MHz]	Throughput (LUT based) [Gbps]	Throughput-to-Area (LUT based) [kbps/Slice]	Throughput-to-Area (GF based) [kbps/Slice]
Spartan 3	14747	53.60	2.49	0.17	0.19
Virtex 4	13798	99.39	4.63	0.34	0.37
Virtex 5	2378	161.66	7.52	3.16	1.32
Virtex 6	2511	158.03	7.36	2.93	1.56

At first glance, it seems that the Groestl still performed poorly on lower-end FPGAs such as Spartan 3 and Virtex 4. However, on Virtex 5 and Virtex 6, Groest's performance became very competitive. The cause for this trend can be understood by looking the resources required by the GF and ROM implementation of S-Box.

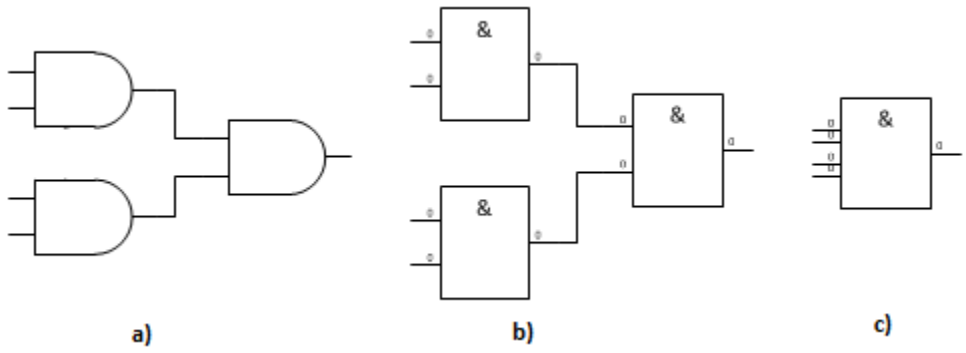

Figure 5.2: Logical implementation verses ROM-based implementation

To implement circuit shown on Figure 5.2a, one can use the scheme shown on Figure 5.2b or Figure 5.2c. The implementation in Figure 5.2b resembles the original logical implantation that utilizes three 2-input LUTs to replace the three AND gates. The implementation in Figure 5.2c resembles more of a ROM-based approach which utilizes a 4-input LUT to implement the entire circuit as a lookup table. Since the implementation on Figure 5.2c uses a 4-input LUT which has the same logic capacity as four 2-input LUTs, one can say implementation on Figure 5.2c is more costly than the implementation on Figure 5.2b. Therefore, any implementation on an FPGA done through logical implementation instead of the ROM-based approach is more efficient. This is the reason why ROM-based approach of implementing the S-Box on Spartan 3 and Virtex 4 is inferior to the GF-based approach.

The reason why Groestl performs better in Virtex 5 and Virtex 6 can be understood by looking at the logical resources used by the different platforms. Spartan 3 and Virtex 4 are relatively old FPGAs that use 4-input LUTs while the Virtex 5 and Virtex 6 use 6-input LUTs. Typically LUTs with higher number of inputs are more powerful because they can implement functions with greater complexity. However, more logic is required to implement a larger LUT. Therefore implementing simple logic using large LUTs is wasteful.

Table 5.4: LUT resources used by a Virtex 5 FPGA to implement the GF based S-Box

Cell Usage	Num
LUT2	2
LUT3	4
LUT4	12
LUT5	9
LUT6	23
Total	50

Looking at the resources needed on the Virtex 5 FPGA, shown in Table 5.4, more than fifty percent of the LUTs needed have less than six inputs. Note that implementing a piece of logic on a LUT6 when a LUT5 is adequate wastes fifty percent of the logical resources on the LUT6. This translates to more than half of

the LUTs being used at fifty percent. Furthermore, around thirty-five percent of the LUTs are using less than twenty-five percent of their logical resources. This indicates that the GF-based S-Box on FPGAs with larger LUTs like Viretx 5 and 6 have a large number of LUTs that are not fully used. This will drive the total amount of resources needed for the implementation to be higher than what is needed if the ROM-based implementation was used. In the case of Spartan 3 and Virtex 4, however, the waste is not as great because they use LUT4 instead of LUT6; thus only twelve percent of the LUTs are being under-utilized.

It can be observed that the speed of the candidate Groestl is slower with the ROM-based implementation of an S-Box on the Spartan 3 but faster on the Virtex 5. The reason cannot be determined at this point; the timing report of the S-Box implemented in both schemes has been looked into and it has been observed that ROM-based implementations are always about two times faster than the GF-based implantations on both Spartan 3 and Virtex 5.

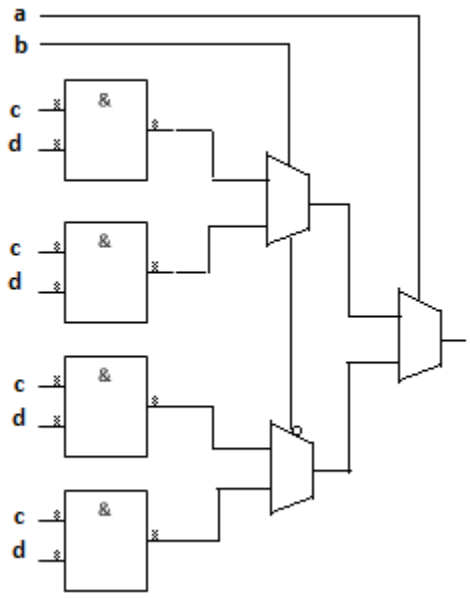

Figure 5.3: Logical diagram of the implementation of a LUT4 using LUT2 cells and MUXs

The speed advantage of ROM-based implementation, shown in Figure 5.3, is due to its circuit construction wherein only one layer of LUTs and few layers of MUXes are being used. The circuit construction of the GF-based implementation, however, requires multiple layers of LUTs and MUXes, thus increasing the length of the critical path. Despite the improvement of the speed of the ROM-based S-Box, it is perhaps the routing congestions due to a large number of extra LUTs on the Spartan 3 that causes the overall candidate performance to drop.

From this experiment, it can be seen that using a single RTL to evaluate a candidate can sometimes lead to inaccurate ranking of the candidates. To see the whole picture, the RTL codes of the candidates may need to be optimized according to their device and platform at the cost of increased manpower.

5.2 Study of the effects of technology scaling

In this section, the question of how the technology node affects the performance of the candidates on ASIC as well as FPGA will be addressed. Typical trends in terms of performance when moving towards a smaller feature size in hardware are that the circuit will operate faster, the circuit will consume less area, and the circuit will dissipate less power. To evaluate the effects in ASIC, all the hash candidates and SHA-2 run through the synthesis process to illustrate the change in performance. Instead of using four design points, the two extreme points at MinArea and MaxSpeed were selected to illustrate the effects. For an FPGA, on the other hand, the results obtained from the previous section will be used.

5.2.1 ASIC

For analysis on ASIC device, the UMC 130nm and UMC 90nm libraries are used. The HDL codes are mapped into primitive gates of those libraries with Synopsys Design Compiler. Typical case for characterization of the standard cell libraries is used. The 90nm technology uses nine metal layers and 180nm technology uses eight metal layers. Usually, having more metal layers allows a denser routing scheme which makes the design more area efficient.

Figure 5.4 and 5.5 show the area and throughput results for 90nm and 130nm technologies on MinArea while Figure 5.6 and 5.7 show the area and throughput results on MaxSpeed. Figure 5.8 and 5.9 show the area and throughput results normalized to SHA-2 at the two design points.

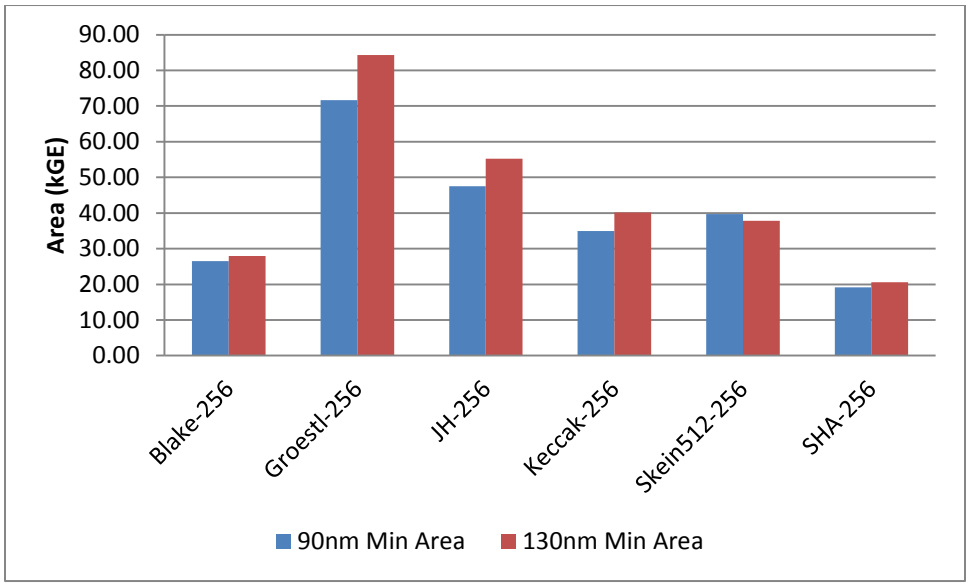

Figure 5.4: Comparison of the area results of SHA-3 candidates with 90nm and 130nm technology at the design point MinArea

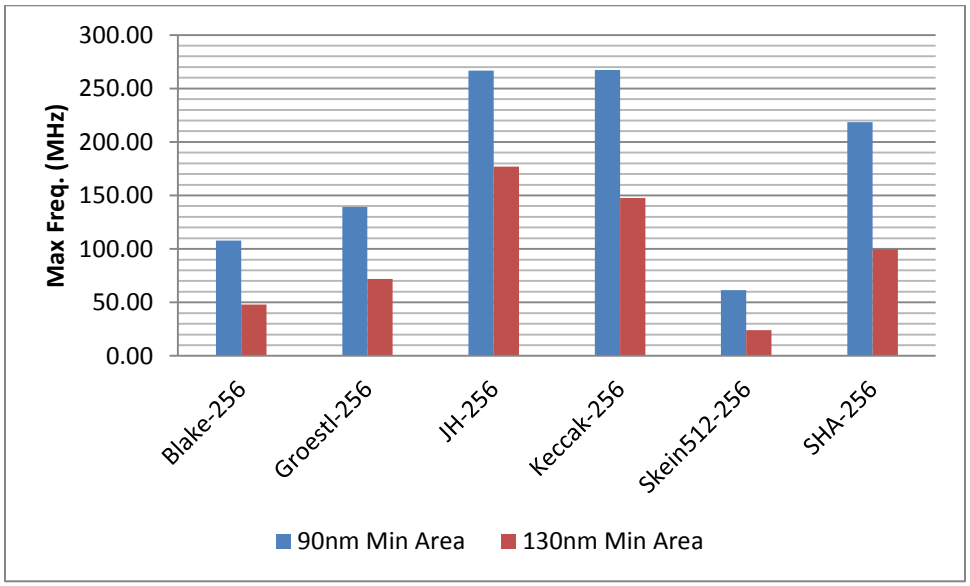

Figure 5.5: Comparison of the maximum frequency results of SHA-3 candidates with 90nm and 130nm technology at the design point MinArea

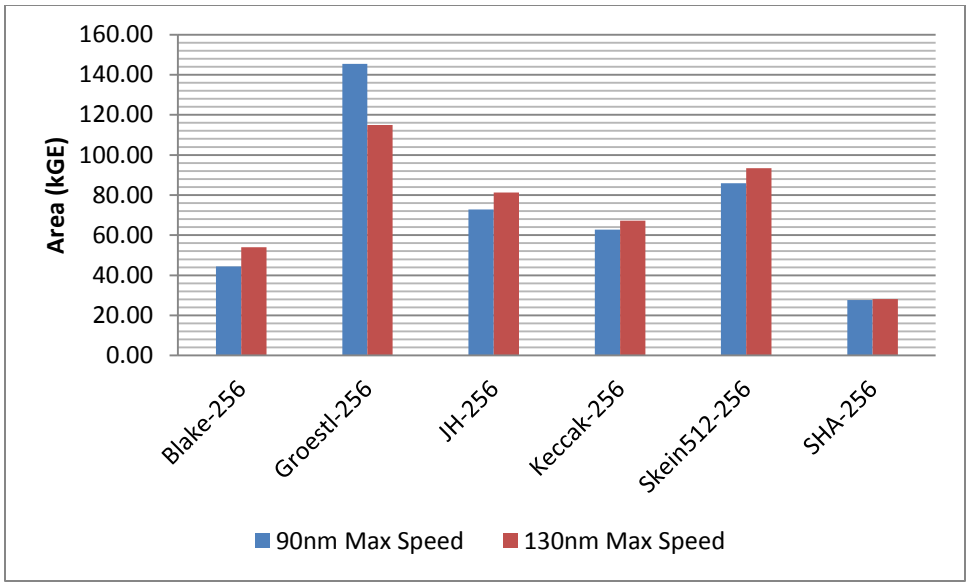

Figure 5.6: Comparison of the area results of SHA-3 candidates with 90nm and 130nm technology at the design point MaxSpeed

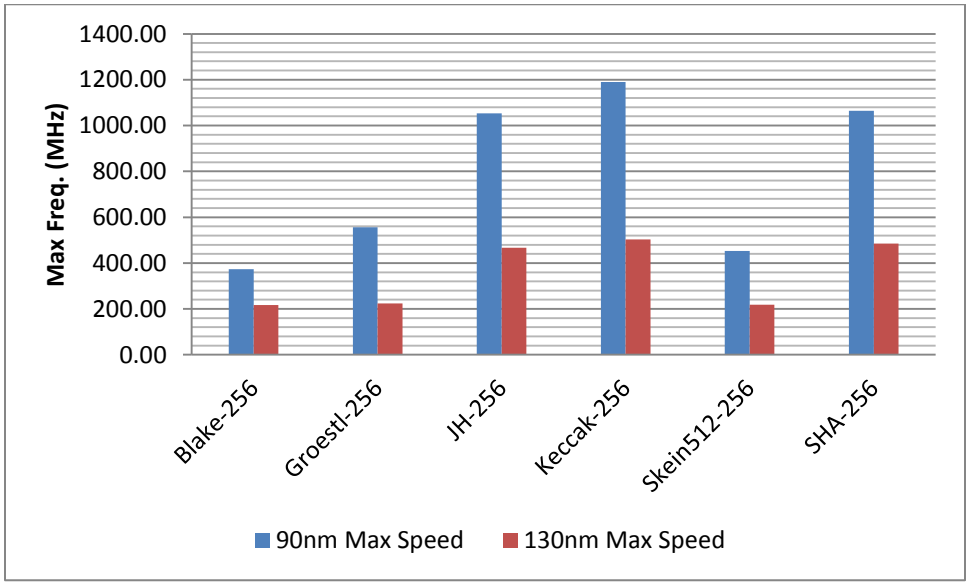

Figure 5.7: Comparison of the maximum frequency results of SHA-3 candidates with 90nm and 130nm technology at the design point MaxSpeed

For area results across different technology nodes, the changes are mostly insignificant. Although, candidates Groestl at MinArea and Skein at MaxSpeed have larger area in 90nm than 130nm technology, this can be explained by possible differences in the libraries at the two technology points. The maximum

frequency results, on the other hand, changed significantly. The maximum frequency of the candidates in 90nm is almost twice as much as their numbers in 130nm.

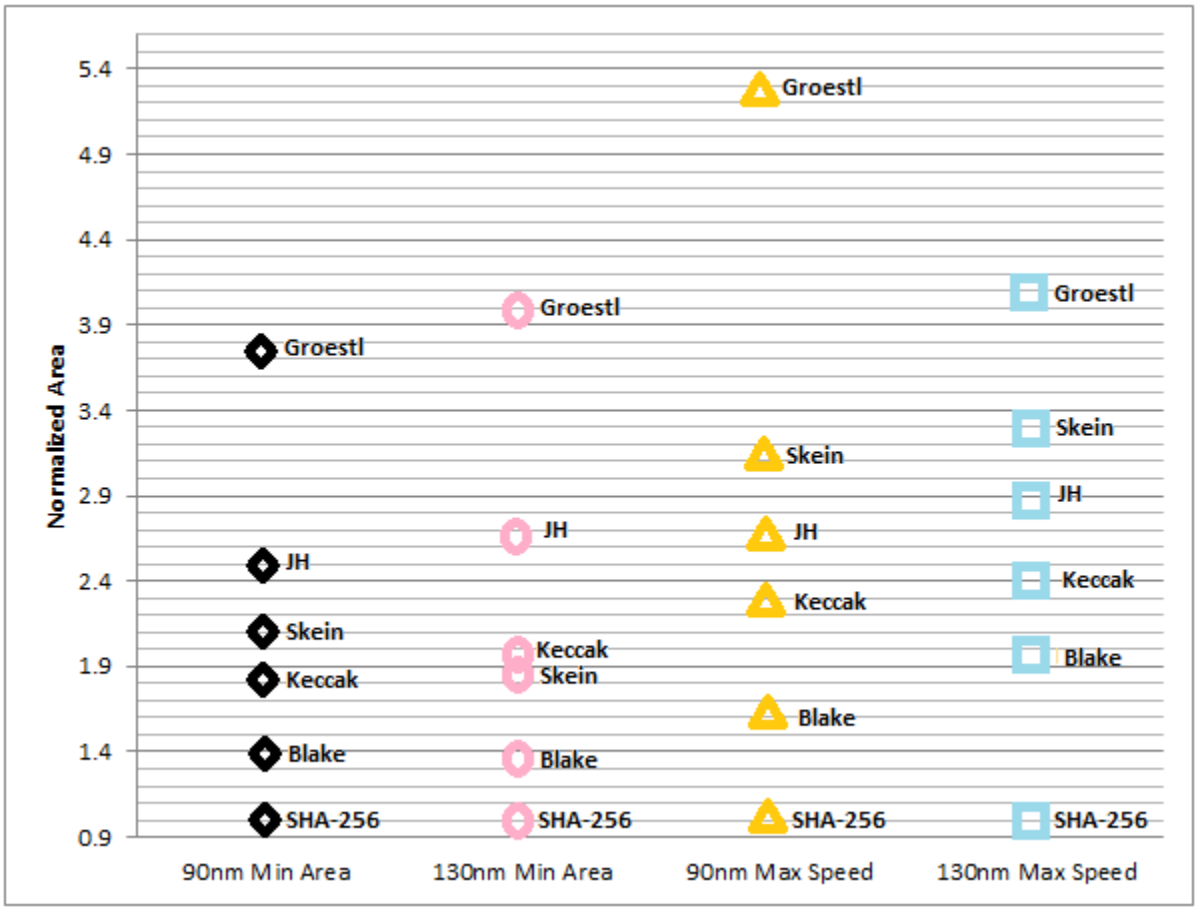

Figure 5.8: Comparison of the area results normalized to SHA-256 of SHA-3 candidates with 90nm and 130nm technology

It can be seen that the ranking in terms of area and throughput are mostly unchanged across different technology nodes. However, the relative area of the candidates changes significantly for some cases from one technology to another. Therefore, the effect of technology scaling on ASIC may not be completely linear as expected. Thus, the results may not hold for more advanced technologies.

In summary, the effects of technology scaling on ASIC are mostly linear with occasional exceptions. These exceptions may change the overall ranking. Therefore, the results can only be described as inconclusive.

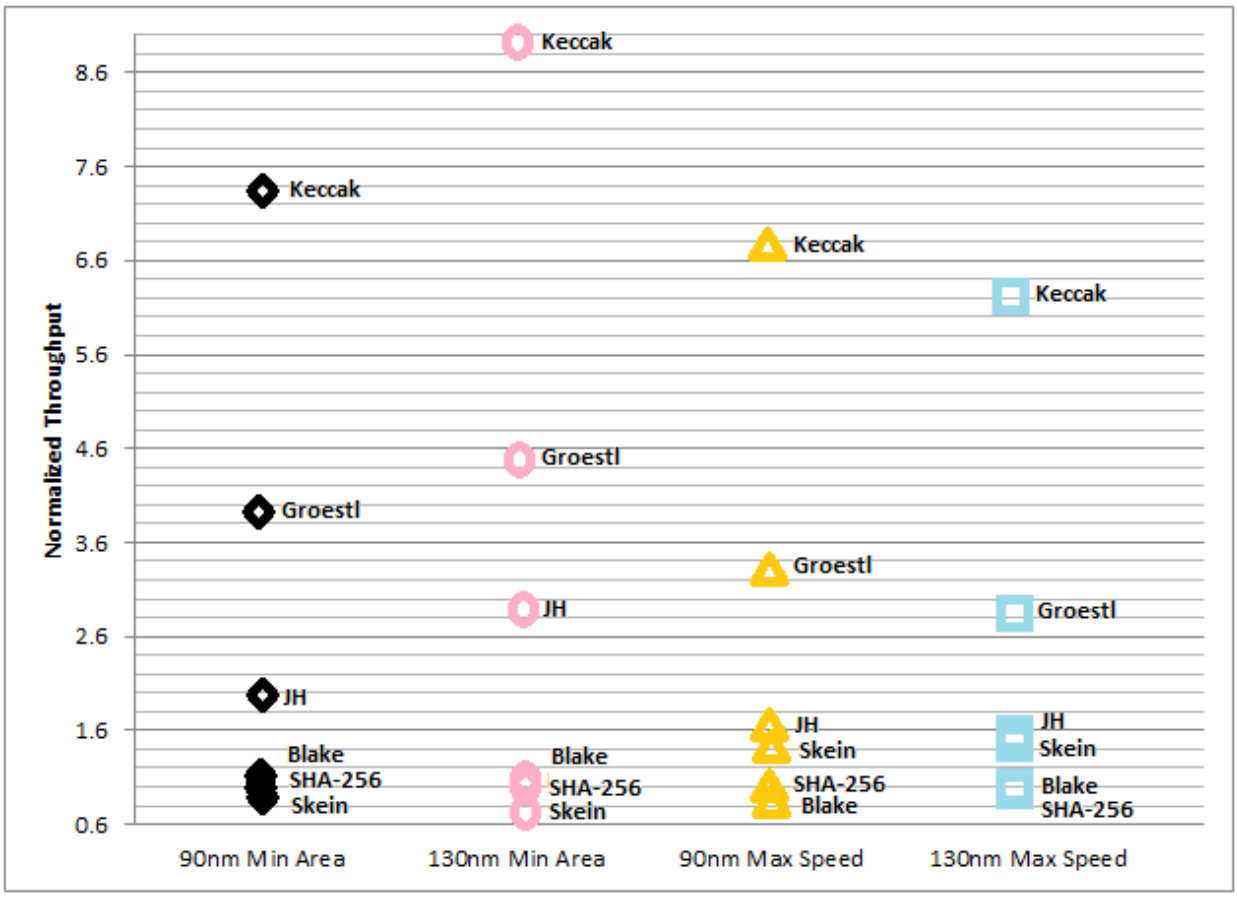

Figure 5.9: Comparison of the throughput results normalized to SHA-256 of SHA-3 candidates with 90nm and 130nm technology

5.2.2 FPGA

For an analysis on FPGA device, the previous results from the ATHENA flow will be used due to its comprehensiveness and consistency. The Xilinx FPGAs used are Spartan 3, Virtex 4, Virtex 5, and Virtex 6 as platforms for evaluation. Their feature sizes are 90nm, 90nm, 65nm, and 40 nm. It should be noted that on FPGA platforms features size is not the only factor that is affects the performance of the design. When moving towards smaller feature sizes, FPGAs typically grow in terms of LUT size and hardware capability.

Table 5.5: Throughput to area ratio results normalized to SHA-256 from AHENA flow

	Spartan 3	Virtex 4	Virtex 5	Virtex 6
Keccak-256	3.27	3.07	2.53	1.23
SHA-256	1.00	1.00	1.00	1.00
JH-256	0.79	0.74	0.74	0.56
Skein512-256	0.72	0.60	0.62	0.40
Groestl-256	0.59	0.55	0.52	0.31
Blake-256	0.51	0.45	0.40	0.26

As already mentioned, the ranking of the candidates remains the same across different platforms. Some may expect that the effect of the technology scaling is at most modest due to the perseverance of the ranking, however, this may not be the case. At Table 5.5, each candidate's performance in terms of throughput-to-area ratio is normalized with respect to SHA-2. It can be observed that significant changes in terms of relative performance can be seen from the table. For example, the candidate Keccak, outperforms the reference algorithm SHA-2 by more than three times on Spartan 3, but is only slightly better on Virtex 6. It is not clear whether this phenomenon is due to the effects of difference in feature size or difference in FPGA architecture.

Figure 5.10 and 5.11 show the area and throughput results for hash candidates across different platforms normalized to SHA-2. With close performances some candidates' numbers occasionally traded places with each other; therefore, the effect of technology scaling on FPGA is not completely uniform either. Therefore, the results may change on other FPGA platforms.

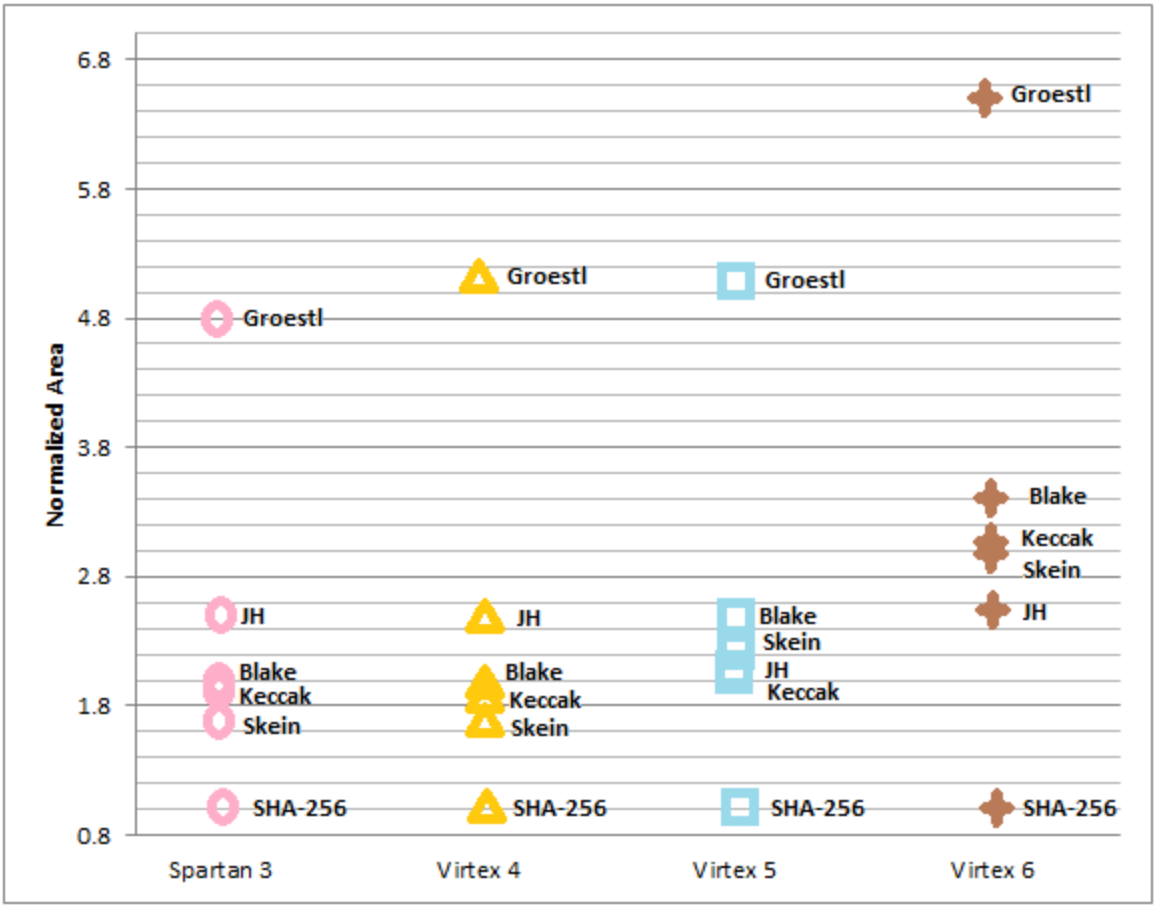

Figure 5.10: Comparison of the area results normalized to SHA-256 with ATHENA flow

In this section, the effects of technology scaling have been seen on the result of hash candidates on ASIC as well as on FPGAs. Those effects are minor in most cases with occasional spikes. Therefore, they are still non-trivial because they may affect the ranking as well as relative performance and cost of the candidates. Thus, for different technologies or platforms, the analysis must be redone in order to accurately measure the performance of each candidate, instead of just extending the current results.

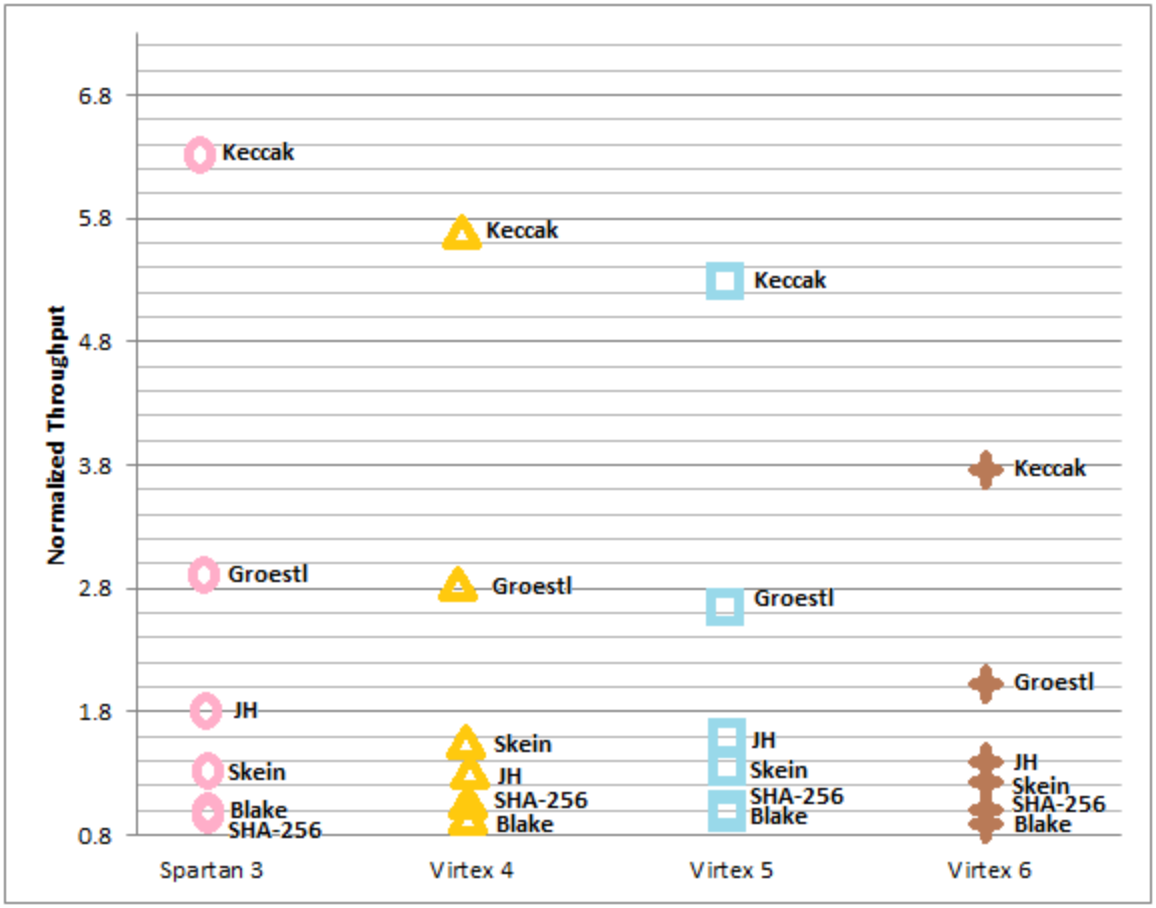

Figure 5.11: Comparison of the throughput results normalized to SHA-256 with ATHENA flow

5.3 Conclusion

In this chapter, additional experiments were discussed. An alternative implementation of Groestl was performed in order to explain why that implementation has better performance results than the original implementation. The effects of technology on ASIC as well as on FPGA was also studied. The effects of technology scaling still may affect the overall ranking of the candidates. Therefore, many other tasks such as RTL optimization and implementation of hash candidates on different FPGA platforms or with different ASIC libraries can be done for a more comprehensive analysis.

Chapter 6

Conclusions and Future Work

In this thesis the motivation behind the hardware evaluation of hash candidates was discussed as well as their definition, applications, and properties. Then, some commonly used operations and modules used by cryptographic hash functions were mentioned as well as the five SHA-3 candidates and the hardware devices with which they will be implemented.

Later, the methodology of the experiments, including the interface, the metrics and technology node chosen were shown. CAD flow was also mentioned in addition to the process followed for ASIC and FPGA implementations and the decisions made during the design process. The reasons behind the four design points picked for the ASIC implementation were explored. After that an overview of the project and the framework of the experiment were given. An overview of the implementation, results, and performance of the candidates on FPGA as well as on ASIC were given. Candidate Keccak outperforms other SHA-3 candidates by at least a factor of two on an FPGA platform as well as on ASIC. The short comings of some candidates were discussed by using their results in area and throughput. Finally, the results across the two FPGA flows and across FPGA and ASIC were compared. The significance of the ATHENA process on the area and throughput results of the candidates was shown.

In chapter four some additional studies were shown. The improved performance of another Groestl implementation was explained with respect to other original implementation. Also, the effects of technology scaling on FPGA and ASIC was discussed.

Overall, the focus of this thesis is to demonstrate the process for the hardware evaluations of SHA-3 candidate.

6.1 Future work

To give a more in-depth analysis of the hash candidate's performance on ASIC and other potential applications a tape-out of an ASIC chip that will come back in June has been made. Additional experiments can be done on the actual chip, such as measuring the actual maximum frequency and power dissipation. Also, in the future, the algorithm's security from the hardware perspective will be evaluated, such as performing the side channel attacks on the candidates. Lastly, additional tasks that can be performed, such as comparing the candidates' performances on other platforms of FPGA, or, implement the candidates with other libraries or technologies for ASIC.

Bibliography

[1] Secure Hashing, April, 2011. http://csrc.nist.gov

[2] T.Dierks, et al. The Transport Layer Security (TLS) Protocol, August 2008. http://tools.ietf.org

[3] T. Ylonen, et al. The Secure Shell (SSH) Transport Layer Protocol, January 2006. http://tools.ietf.org

[4] B. Ramsdell, et al. Secure/Multipurpose Internet Mail Extensions, July 2004. http://tools.ietf.org

[5] S. Kent , et al. Security Architecture for the Internet Protocol, November 1998. http://tools.ietf.org

[6] J. Callas, et al. OpenPGP Message Format, November 1998**.** http://tools.ietf.org

[7] NIST Comments on Cryptanalytic Attacks on SHA-1, January 2009. http://csrc.nist.gov

[8] Intel® Advanced Encryption Standard (AES) Instructions Set, March 2011. http://software.intel.com

[9] W. Stalling. "Hash Functions," Cryptography and Network Security Principles and Practices, Prentice Hall, pp 334-338, 2009

[10] X.Y Wang, et al. Finding collisions in the full SHA-1, Crypto 2005

[11] Cryptographic Hash Algorithm Competition, December, 2010. http://csrc.nist.gov

[12] Tentative Timeline of the Development of New Hash Functions, December 2010. http://csrc.nist.gov

[13] C. Paar, et al. Understanding Cryptography, Springer 2010

[14] F.R. Henriquez, et al. Cryptographic Algorithms on Reconfigurable hardware, 2007 Springer

[15] K. Gaj, et al. Comprehensive Comparison of Hardware Performance of Fourteen Round 2 SHA-3 Candidates with 512-bit Outputs Using Field Programmable Gate Arrays. NIST 2nd SHA-3 Candidate Conference, 2010.

[16] B. Baldwin, et al. FPGA Implementations of the Round Two SHA-3 Candidates, NIST 2nd SHA-3 Candidate Conference, 2010.

[17] K. Kobayashi, et al. A Prototyping Platform for Performance Evaluation of SHA-3 Candidates. Proceedings of HOST2010, 2010.

[18] Z. Chen, et al. A Hardware Interface for Hashing Algorithms. IACR ePrint archive, 2008/529, 2008.

[19] K.Gaj, et al. Fair and comprehensive methodology fir comparing hardware performance for fourteen round two SHA-3 candidates using FPGA. Proceedings of CHES2010, LNCS Springer, 2010.

[20] Kris Gaj, et al. ATHENa – Automated Tool for Hardware EvaluatioN: Toward Fair and Comprehensive Benchmarking of Cryptographic Hardware using FPGAs. 20th International Conference on Field Programmable Logic and Applications, FPL 2010.

[21] SHA-3 Hardware Implementations. April 2011, http://ehash.iaik.tugraz.at

[22] P. Gauravaram, et al. Groestl – a SHA-3 candidate, January 2011. Ecrypt. http://ehash.iaik.tugraz.at