IMPLEMENTATION & EVALUATION OF LOW AREA AND HIGH SPEED DESIGN ARCHITECTURES OF SHA-3 CANDIDATE BLAKE ON AN FPGA PLATFORM

By

Muhammad Arsalan

2010-NUST-MS PHD- Elec (Comm-N)-14

Thesis Submitted in Partial Fulfillment of the Requirement for the Degree of Master of Science in Electrical Engineering With specialization in Communication

Thesis Supervisor

Dr. Arshad Aziz

Department of Electrical & Power Engineering,

Pakistan Navy Engineering College,

National University of Sciences & Technology (NUST),

H-12 Islamabad, Pakistan

December 2012

In the name of Allah, the most Beneficent and the most Merciful

DECLARATION

I hereby declare that I have developed this thesis completely on the basis of my delicate efforts under the sincere guidance of my supervisor Dr. Arshad Aziz. All sources used in this work have been cited and the contents of this thesis have not been plagiarized. No portion of the exertion presented in this thesis has been submitted in support of any application for any other degree of qualification to this or any other university or institute of learning.

_____

Muhammad Arsalan

ACKNOWLEDGEMENTS

First of all praise is to Allah, the Almighty, on whom ultimately all of us depend for sustenance and guidance. Secondly, I would like to show my sincere appreciation to my supervisor Dr. Arshad Aziz for his support, incessant guidance, meticulous suggestions and astute criticism during the practical phase and for his inexhaustible patience during the correction phase of this dissertation. His timely and efficient involvement helped me a lot to shape this thesis into its final form and I want to express my sincerest appreciation for his support in any way that I may have asked.

I wish to broaden my tremendous gratitude and thanks to my GEC Members Prof. Dr. Pervez Akhtar, Dr. Sameer Hashmat Qazi and Dr. Khawaja Bilal Mahmood for their support, assistance and guidance. I would also like to express my gratitude to Head of Department Dr. Ataullah Memon, for his continuous support, precious suggestions and assistance, especially for the provision of all kinds of facilities during my thesis work.

Last but not least, I wish to express my sincere thanks to all those who have one way or another helped me in making this research a success.

DEDICATIONS

I dedicate my dissertation work to my family, my teachers and many friends especially I want to express my feeling of gratitude to my loving parents for their kind support.

ABSTRACT

Cryptographic hash functions are used to protect information integrity and authenticity in a wide range of applications. Serious attacks have been reported in recent years against cryptographic hash algorithm, including SHA-1, and SHA-2 family, which is currently used as standard, developed by National Institute of Standards and Technology (NIST).Therefore, NIST has decided to issue a Call for a New Cryptographic Hash Algorithm (SHA-3) Family. Five Secure Hash Algorithms were selected after Round-3 of SHA-3 competition. A basic evaluation criterion for selection SHA-3 standard is based on the strength of security and efficiency in Hardware implementation for wide verity of platforms. Blake is one of the candidates of round three of SHA-3 competition. Performance evaluation of Blake algorithm on FPGA is depends on the type of architecture implemented. The core of the design is based on compression function, which includes G-Function calculation. Due to the symmetric nature of compression function of Blake Algorithm, different design architectures can be implied based on serialization of common procedures. These design variations resulted in low to high area and speed results.

In this research work, we have implemented three design architectures of Blake algorithms on FPGA based on number of G-Functions execution at a time. The initialization and finalization step are same for all designs and variation is based only on round function execution. In the first design we have used 8 G-Functions in parallel, second design used hardware of 4 G-Functions and the third design is based on 2 half-G Functions in parallel with pipeline registers between them. Pros and cons of these architectures are discussed in this thesis. In fully autonomous design implementation, common I/O interface is provided and only Slice resources of FPGA are used for design implementation for fair comparison. For tradeoff analysis, each design is implemented using three different optimization strategies, these are “Area”, “Speed” and “Balance” design approaches. We have implemented both Blake-256 and Blake-512 designs on Virtex 5 FPGA. For Blake-256, maximum throughput of 2.6 Gbps is achieved for 8G design. 1G design gives most efficient results in terms of number of slices utilized i.e. 416 slices. The optimized delay path is utilized in 4G design with respect to Virtex 5 architecture. That’s gives maximum TPA of 2.1. On the other hand, Blake-512 design implementation gives throughput of 4.7 Gbps when applying 8G design. 4G design gives the best TPA of 1.904 and least area of 801 slices resulted when 1G design is implemented. The evaluation concluded that 4G design gives

best results in terms of TPA. Overall research suggests that selection of architecture is dependent upon type of application either high speed requirements or low area constraints, suitable optimization could be performed in a particular domain to achieve best design results.

Table of Contents

DECLARATION i
ACKNOWLEDGEMENTS ii
DEDICATIONS iii
ABSTRACT iv
CHAPTER 1: INTRODUCTION 1
1.1 Motivation 2
1.2 Research Goals and Objective 2
1.3 Approach 3
1.4 Organization 4
1.5 Summary 4
CHAPTER 2: BLAKE ALGORITHM 5
2.1 Hash Functions 5
2.2 Applications of Hash Functions 5
2.3 Standard Hash Functions 6
2.4 NIST Competition for SHA-3 7
2.5 Blake SHA-3 Candidate 7
2.6 Blake-256 Algorithm IO Interfaces 8
2.7 Blake Architecture 9
2.8 Padding 10
2.9 Counter 10
2.10 Initialization 10
2.11 Round Function 11
2.12 Permutation Table 12
2.13 G-Function 13
2.14 Finalization 14
2.15 Summary 15
CHAPTER 3: BACKGROUND AND LITERATURE REVIEW 16
3.1 Design Methodologies 16
3.2 Related Work 18
3.3 Summary 20

CHAPTER 4: DESIGN APPROACH 22
4.1 BLAKE Implementation on FPGA 22
4.2 Performance Evaluation 22
4.3 Performance Metrics 23
4.4 Common Design Features 23
4.4.1 Wrapper Design 24
4.4.2 Message/Constant Select Mechanism 24
4.5 8G Design 25
4.6 4G Design 27
4.7 1G Design 29
4.8 Summary 31
CHAPTER 5: IMPLEMENTATION AND RESULTS 33
5.1 Optimization Strategies 33
5.2 Post Place and Route Implementation Results 33
5.3 Timing Comparison 36
5.4 Area Utilization Comparison 37
5.5 Throughput per Area 38
5.6 Summary 40
CHAPTER 6: CONCLUSION AND FUTURE RECOMMENDATIONS 41
6.1 Conclusion 41
6.2 Future Recommendations 41
6.3 Publications 42
BIBLIOGRAPHY 43

List of Figures

Figure 2.1 Password Authentication example 6
Figure 2.2 Blake-256 Algorithm IO Interfaces 8
Figure 2.3 Blake algorithm autonomous design architecture 9
Figure 2.4 Internal State Initialization 11
Figure 2.5 Round Function (a) Column Step (b) Diagonal Step 11
Figure 2.6 G-Function Architecture 13
Figure 4.1 Blake-256 wrapper architecture 24
Figure 4.2 Message and Constant Select Mechanism 25
Figure 4.3 8G Design Architecture 26
Figure 4.4 8G Design message and constant selection 27
Figure 4.5 4G Design Architecture 28
Figure 4.6 1G Architecture Design 29
Figure 4.7 1G Design Flow (1 Round) 30
Figure 4.8 1G Design message and constant selection 31
Figure 5.1 Timing Comparison 37
Figure 5.2 Resource Utilization Comparison 38
Figure 5.3 Throughput/Area Comparison for Blake-256 39
Figure 5.4 Throughput/Area Comparison for Blake-512 39

List of Tables

Table 2.1 SHA Algorithms Evolution 6
Table 2.2 Permutation Table 12
Table 2.3 Blake-256 Constants 13
Table 2.4 Blake-256 Initialization Value 14
Table 2.5 Finalization (Hash Value calculation) 14
Table 3.1 Blake Implementation Results 20
Table 5.1 Design optimization strategies 33
Table 5.2 BLAKE-256 Post place and route implementation results on Virtex 5 FPGA 34
Table 5.3 BLAKE-512 Post place and route implementation results on Virtex 5 FPGA 34
Table 5.4 BLAKE-256 Place and route results comparison (VIRTEX 5) 35
Table 5.5 BLAKE-512 Place and route results comparison (VIRTEX 5) 36

CHAPTER 1: INTRODUCTION

The aim of cryptography is to secure information so that only the intentional parties can read the data. Cryptography is deployed with cryptographic algorithms. These are based on mathematical functions used for converting messages into a form that cannot be retrieved by formal means. Cryptosystems today are used in many areas like banking systems, identification systems and entertainment systems using encrypted storages and even in electronic car locks. There are mainly three types of cryptography algorithms. These are secret key, public key, and hash functions. Unlike secret key and public key algorithms, hash functions called as one-way encryption and there is no key involved in Hash Function calculation.

A secure hash function is a cryptographic primitive. It is a transformation in which variable-size input is taken and fixed-size string is returned, which is called as hash value. A fixed-length hash value is computed based on the plaintext that makes it impossible for either the contents or length of the plaintext to be recovered [1].The prime application of hash functions in cryptography is message integrity. The hash value provides a digital fingerprint of a message, which ensures that the message has not been altered by an intruder, virus, or by any other means. Other applications of Hash Functions are message authentication codes, digital signatures and password protection. [2]

Secure Hash Algorithm (SHA) family of hash functions developed by the National Institute of Standards and Technology (NIST) is currently used as standard for Hash Functions [3]. Serious attacks have been reported in recent years against cryptographic hash algorithm, including SHA-1, and SHA-2 family[4][5]. NIST has decided to standardize new hash algorithm to augment the ones currently specified in FIPS 180-2 [5]. NIST issued a Call for a New Cryptographic Hash Algorithm (SHA-3) Family in a Federal Register Notice on Nov. 2, 2007[6].In this regard 64 entries were submitted in which only 14 were selected for the second round of the competition.
Based on the public feedback and internal reviews of the second-round candidates, NIST selected five SHA-3 finalists - BLAKE, Grøstl, JH, Keccak, and Skein to advance to the third and final round of the competition on December 9, 2010. The result of the contest is expected to be announced at the end of 2012[7].

1.1 Motivation

Blake is one of the candidates of round three of SHA-3 competition [7]. Blake algorithm consists of family of four hash functions BLAKE-224, BLAKE-256, BLAKE-384 and BLAKE-512. A basic evaluation criterion for selection SHA-3 standard is based on the strength of security and efficiency in Hardware implementation for wide verity of platforms [7].

Due to reconfigurable characteristic of FPGA, sufficient logic and storage elements availability, complex algorithms can be built on FPGA platform. Therefore, FPGAs are widely used for evaluation of Hash Function algorithms. In order to implement the same algorithm on FPGA, different architectures could be implied to achieve different design objectives. One of the most important issues is the cost-performance tradeoffs. Cost in FPGA hardware design can be interpreted in different ways like logic area in terms of slices, memory elements in terms of Block RAMs and Distributed RAMs. Another commonly concern faced in hardware design is the parallel and serial design tradeoff.

One way to implement the design in which different procedures executed in parallel to complete the algorithm in just one or few cycles, on the other hand pipeline architectures use many cycles under a fast clock to complete the algorithm. The facility provided by FPGA, to implement different types of design architectures to find most efficient design in terms of low area and high speed performances and capability of Blake algorithm to execute for variety of design architectures motivate us to work in this area.

1.2 Research Goals and Objective

The objective of this research work is to analyze various design architectures of Blake algorithm best suited for FPGA device architectures. Evaluation of three types of design architectures based on High Speed, Low Area and Highest TPA (Throughput per Area) is goal of this research. Various pros and cons of all architectures have discussed to conclude the performance evaluation of each design. Testing and simulation of all design is performed using test vectors given in design specification document of Blake and output results are compared with the results given in specification document for verification.

Design implementation is focused on FPGA Device architecture. Two versions of Blake algorithm based on internal data bus have been implemented on FPGA. These are Blake-256 and Blake-512. Performance evaluation each design has been performed to find out most efficient design results. Objective of this work is to find optimized architecture in three areas for Blake algorithms, one for highest throughput, second for resource constrain environment and final which gives most efficient results in terms of Throughput per Area on Xilinx Virtex 5 FPGA.

1.3 Approach

In this research work, we have implemented three design architectures of Blake algorithms based on number of G-Functions execution at a time. The initialization and finalization step are same for all designs and variation is based only on round function execution. In the first design we have used 8 G-Functions in parallel, second design used hardware of 4 G-Functions and the third design is based on 2 half-G Functions in parallel with pipeline registers between them. In fully autonomous design implementation, common I/O interface is provided and only Slice resources of FPGA are used for design implementation for fair comparison. Mechanism for selection of message and constant values for each round is also discussed for all three architectures.

Verilog HDL is used for designing of Blake algorithms. Two versions of this algorithm based on internal data bus are implemented on FPGA. These are Blake-256 which generates 256-bits Hash value for 512-bits input message and Blake-512 which generates 512-bits Hash value for 1024-bits messages value. Internal 64-bits wide bus is used in Blake-512 while internal word size of Blake-512 architecture is 64-bits wide.

Design for three architectures for both versions is implemented on Xilinx Virtex 5 xc5vlx-110t FPGA[8]. Selection is based on the fact that most of the evaluations work for SHA-3 candidate has been performed on these devices. Blake algorithm architecture is design to optimize the performance based on Slice architecture of Virtex 5.

For synthesis and implementation purpose, Xilinx ISE tool is used. Tradeoff analysis is performed by implementing each design using three different optimization strategies. These are “Area optimization without IOB packing”, “Timing Performance with physical synthesis” and “balanced design approach” this is default in ISE Tool. Testing of designs consists of functional and timing simulation i.e. post-route simulation will be performed to verify the design results.

1.4 Organization

Rest of the thesis is organized as follows. In chapter 2, detail design specification of Blake algorithm is discussed with the background of Hash Functions and Blake algorithm emergence for NIST competition. Chapter 3 thoroughly describes the different design methodologies for implementation of Blake algorithm on FPGA performed by various researchers so far. Analysis of different design approaches used by other researchers related to the both compact and high throughput designs is primary focus of this chapter. Chapter 4 describes our design methodology in detail. It describes detail architecture, common interfaces and performance metrics along with wrapper design of three proposed architectures of 8G, 4G and 1G design. In chapter 5, Post-Route implementation results are given. Comparison tables and graphs are shown to analyze and compare various design approaches with respect to device platform used for both Blake-256 and Blake-512 versions. Comparison of our proposed design results with other designs is also given in this chapter. Finally in Chapter 6 conclusion is drawn for this research work with future recommendations and outcomes of overall research phase.

1.5 Summary

In this chapter we have discussed about basic introduction to cryptography, hash functions and brief summary of the overall thesis work. The aim of cryptography is to secure information. Hash Functions are one of the three types of Cryptography. It is a transformation of variable-size and fixed-size Hash Value string is returned. SHA-2 Hash algorithm is current standard for Hash Function. Due to currently revealed weakness in SHA-2,NIST announced a competition for SHA-3 for selecting next Hash Function algorithm on November 2007. Five finalists are selected for Round-3 of this competition. In this research work, Blake algorithm included in these finalists, is implemented on FPGA hardware. Motivation behind its implementation is the fast evaluation of design algorithm on reconfigurable FPGA and the symmetric nature of Blake The objective of this research work is to find out best design architectures of Blake-256 and Blake-512 with respect to ‘Area’, ‘Speed’ and ‘Balanced’ design objectives. The approach used to meet the research goals is based on number of G-Function execution at a time. Finally, organization of this thesis report given in this chapter that briefly described the contents of each chapter. In next chapter, we will thoroughly discuss the Blake algorithm and its architecture details.

CHAPTER 2: BLAKE ALGORITHM

2.1 Hash Functions

A hash function is a cryptographic algorithm that takes an arbitrary block of data and returns a bit string of fixed length, the hash value, such that a deliberate change to the data with very high probability will modifies the hash value. The encoded data is called as "message," and the hash value is called as message digests or simply digests.

2.2 Applications of Hash Functions

Cryptographic hash functions are very important for many security applications, especially for the authentication related applications, such as message authentication codes, digital signatures and password protection. Digital fingerprint of a message ensures that the message has not been altered by an interloper, virus, or by any other means [1].

For example, in modern operating systems user ask to enter a password that which will be store on hard disk. Next time when user accesses the account, password will be given by the user, the entered password will be compared with the previously stored password and access will be granted. If the password is stored in the form of plaintext in hard-drive, security problem may be aroused because someone could remove the hard disk and extract user’s password data from it. The solution of the problem is to use a cryptographic hash function as shown in Figure 2.1. When the user enters the password, its Hash value is calculated using Hash Function. This Hash value is used for reference in future authentication process. Instead of actual password, its encrypted value is stored in Hard disk rather than its actual form. Next time when user enter password, Hash value is generated again and this Hash value is compared with reference Hash value already stored in hard disk. Using this way if some unauthorized person access the hard disk data, he could not get access to the user’s password [9].

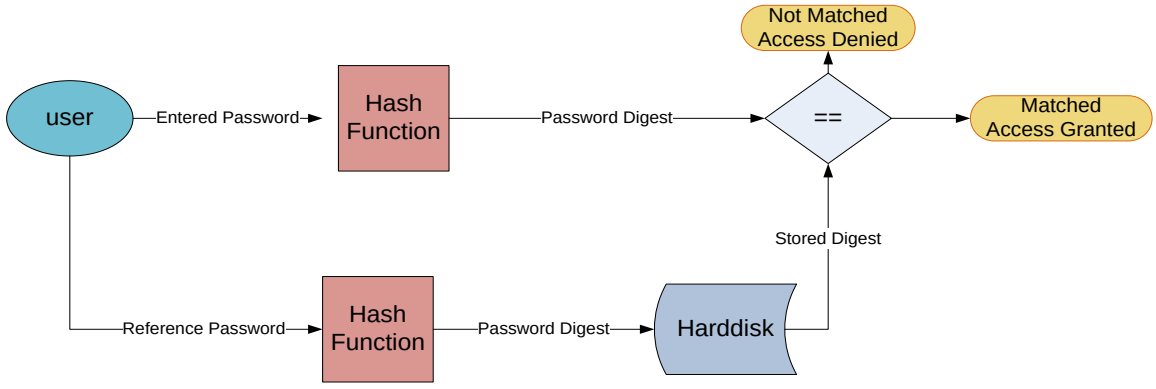

Figure 2.1 Password Authentication example

2.3 Standard Hash Functions

Secure Hash Algorithm (SHA) is a group of hash functions published by the NIST as a US Federal Information Standard [3]. Brief description of each SHA algorithm evolution with time is given in Table 2.1.

Table 2.1 SHA Algorithms Evolution

Standard	Description
SHA-0	Based on 160-bit hash function. It was published in 1993. It was quickly withdrawn due to an unrevealed flaw and replaced by SHA-1 later.
SHA-1	Based on 160-bit hash function, developed by the National Security Agency. Published by NIST in 1995.SHA-1 is very similar to SHA-0. In February 2005, an attack by et. al. was announced [4]. The attacks can find collisions in the full version of SHA-1, requiring fewer than 269 operations.
SHA-2	Four additional hash functions in the SHA family were published by NIST: SHA-224, SHA-256, SHA-384, and SHA-512, collectively known as SHA-2. The 224-bit and 384-bit versions of SHA-2 are simply the 256-bit and 512-bit versions with truncated outputs. The algorithms were first published in 2001 in the draft FIPS PUB 180-2.
SHA-3	New standard for Secure Hash Algorithm is still under development. The algorithm will be selected by choosing different algorithms to a public competition [7]. The final decision is expected to be announced in 2012.

2.4 NIST Competition for SHA-3

Due to substantial weakness found in SHA-1 standard in 2005[4][5]and possible future attacks on SHA-2, NIST decided that it was discreet to develop new hash functions to replace SHA-2 standard. New algorithm will be developed through a public competition announced by NIST. The competition was open for any person or organization. The initial information of the competition, evaluation criteria and its minimum requirements was published in 2007[6]. For evaluation of the algorithm all the candidates had been submitted to NIST by the end of 2008. Among 64 algorithms, 51 submissions qualified for the first round of SHA-3 competition [7].

In the year 2009, the First Hash Function Candidate Conference was hosted by NIST and qualified candidates advanced to the second round were announced. Second Hash Function Candidate Conference was held in 2010 by NIST and the winners of the second round were announced. Five selected candidates among the 14 competitors were advanced into the third round of SHA-3.At this stage, the evaluation criteria had been extended to the hardware domain. Winner of the SHA-3 competition will be announced later in 2012 and selected as a new standard for Hash Function named as “SHA-3” [7].

2.5 Blake SHA-3 Candidate

Blake algorithm is one of the five finalists of Round-3 of NIST SHA-3 contest [7]. Construction of Blake algorithm is based on Hash Iterative Framework (HAIFA) considering its iteration mode[10]. HAIFA produces digests value which is not only based upon input message and internal state of the algorithm, just like Merkle-Damgard construction [11], but also depends on the number of processed bits, recorded by a counter. BLAKE is a family off our hash functions, namely BLAKE-128, BLAKE-256, BLAKE-448, and BLAKE-512. In this research, we have focused on BLAKE-256 and Blake-512. The main difference is in the length of words and in some constants involved in the algorithm.

The Blake hash function proposal for SHA-3 was submitted to NIST by Jean-Philippe Aumasson, Luca Henzen, Willi Meier and Raphael C.-W. Phan [12]. The original submission was made in October 2008 and a revision with some tweaks for the final round of the SHA-3 competition was submitted in January 2011. The names of the modified versions of Blake after

Round-3 are changed to BLAKE-224, BLAKE-256, BLAKE-384 and BLAKE-512. In the subsequent sections, we have described the architecture of Blake-256 in details since, only difference is in size of data words and other architecture details are similar. Another difference is number of rounds which is 14 for BLAKE-128 and BLAKE-256, and 16 for BLAKE-448 and BLAKE-512.

2.6 Blake-256Algorithm IO Interfaces

IO interfaces of Blake-256 are shown in Figure 2.2. The main input to the hash function is the message block input and the main output is the 256-bits digest or hash value. The message block input takes in a 512-bit message. Other input signals are salt and counter, one of which is 512-bits optional.

Figure 2.2 Blake-256 Algorithm IO Interfaces

The salt input is an optional input that the user may utilize to introduce a user-controlled parameter to compression of each message block. The salt is useful for randomized hashing. If the salt is not used, its value is simply set to 0. The counter input adds an extra security layer to the hash function [12].

Blake works in an iterative manner, the hash code of the previous message block is feedback to the input of the hash function for the compression of the next message block. It follows the Hash Iterative Framework (HAIFA) method proposed by Biham and Dunkelman[10]. HAIFA is an improved version of the MD construction [11]. The main ideas behind HAIFA are the

introduction of the counter, a special initial value (IV) for each digest size and a salt to the input of the compression function.

Blake's internal structure is called a local-wide pipe structure in which the internal state of the hash function is much larger than the hash value. In local-wide pipe, the large internal state is initialized from an initial value (or chain value), a salt and a counter. The advantage of the local-wide pipe structure is that it makes ‘local’ collisions impossible [12]. Blake's compression function makes use of Daniel J Bernstein's stream cipher named "chacha"[13]. Blake's compression function is actually a modified version of “chacha”. The designers of Blake added the input of a message word and a constant to the original chacha function. Chacha's performance was found to be excellent during the analysis and it was found to be strongly parallelizable [12].

2.7 Blake Architecture

Figure 2.3 shows fully autonomous design architecture of Blake algorithm. Overall algorithm is divided into three stages; Initialization, Round Function and Finalization.

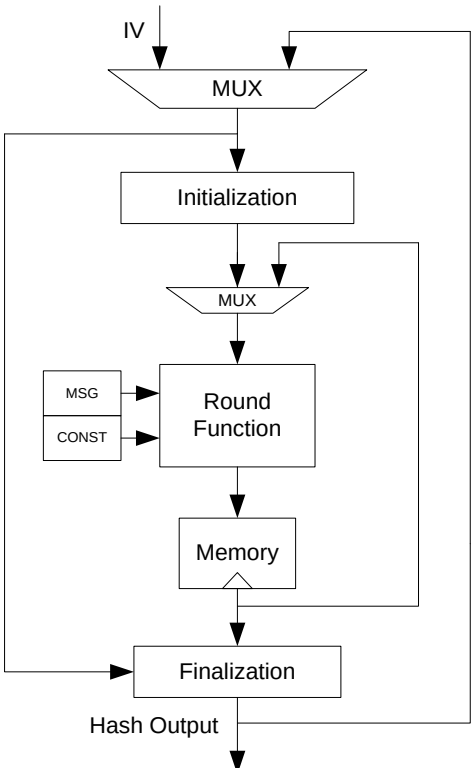

Figure 2.3 Blake algorithm autonomous design architecture

2.8 Padding

A message is padded before hashing. First of all, the message is padded so that it can be divided into an integral number of blocks [12]. For example, if a message has 602 bits, without padding the first 512 bits of the message can be placed in a block but then 100 bits are left 'hanging'. The message is padded to make it a 1024 bit message. Now it can be divided into 2 blocks. Padding is done by appending a number of zero and one bits at the end of the message.

2.9 Counter

The counter is an input that arrives from the HAIFA iteration mode specification. Compression functions have to be iterated such as randomized hashing and the enveloped Merkle-Damgard construction can all be instantiated as part of HAIFA. The counter is based on the sum of the number of message bits that have been hashed so far and the number of message bits in the current block to be hashed. HAIFA makes the compression of each block a function of the counter [12]. For example, we have a message with 1010 bits. After padding, this will be broken into 3 blocks of 512, 498 and 0 message bits. The counter value for the first block is 512; for the second block it is 512+498 = 1010. For the third block, the counter value is set 17 to 0 (irrespective of the number of bits that were previously hashed). Thus, for the third block in our example, the counter value is set to 0.

2.10 Initialization

The internal state of Blake has a local-wide pipe structure. The state is a 4x4 matrix of 32 bit state variables for Blake-256. Thus the state has 16 32-bit variables (or words). The internal state is a core component of the Blake-256 hash function. The inputs (apart from the message block) are used to determine the initial value of the state variables. The initialization of some state variables is done by simply assigning the corresponding value of the initial or chain value to the state variables[12]. For some other state variables, the initialization is shown in Figure 2.4. The initial values h1… h7 is given in Table2.4.

$V_0 = h_0$	$V_1 = h_1$	$V_2 = h_2$	$V_3 = h_3$
$V_4 = h_4$	$V_5 = h_5$	$V_6 = h_6$	$V_7 = h_7$
$V_8 = s_0 \; xor \; c_0$	$V_9 = s_1 \; xor \; c_1$	$V_{10} = s_2 \; xor \; c_2$	$V_{11} = s_3 \; xor \; c_3$
$V_{12} = t_0 \; xor \; c_4$	$V_{13} = t_0 \; xor \; c_5$	$V_{14} = t_1 \; xor \; c_5$	$V_{15} = t_1 \; xor \; c_7$

Figure 2.4 Internal State Initialization [12]

2.11 Round Function

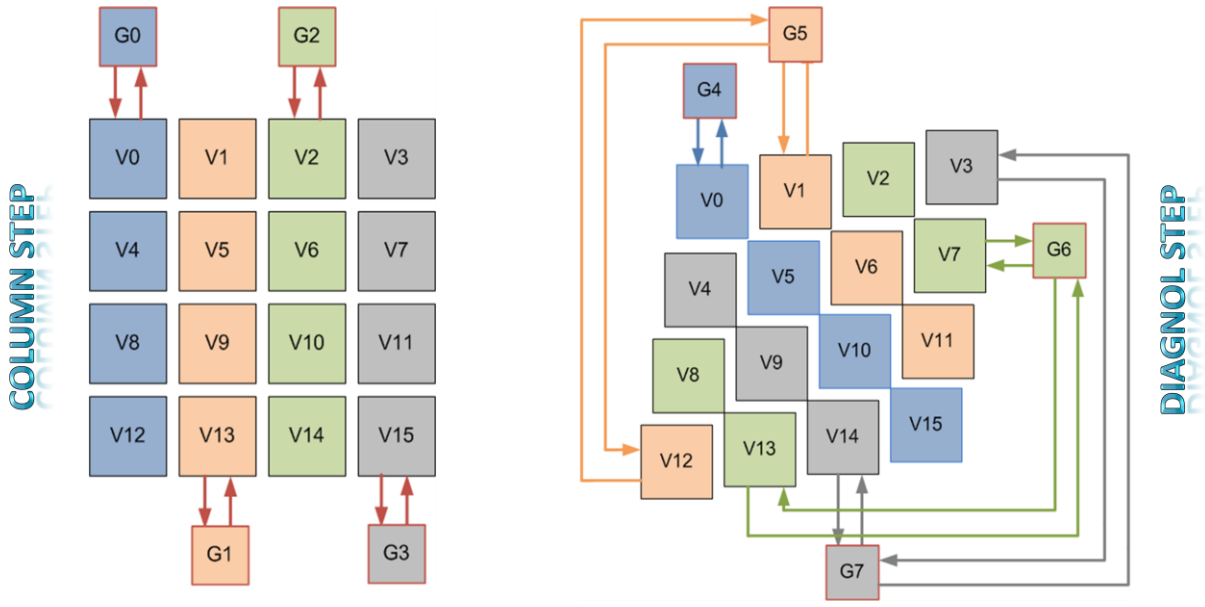

Figure 2.5 Round Function (a) Column Step (b) Diagonal Step [12]

Once it has been initialized, the algorithm of Blake updates the internal by performing some operations on the state. These operations performed on the state change the values of the state variables. The state update employs words of input message block performed by a compression function. It involves operations like addition, rotations and XORing. The compression function perform these operations is called the G-function. Eight G-Functions are calculated in each iteration i.e. one round. Since the state is a 4x4 matrix; it has 4 rows and 4 columns. Each round of a state update can be broken down into the update of the state's columns and diagonals. Column step is shown in left while diagonal step is shown right of figure. The state columns are first updated starting from the first column on the left as indicated by the index of the G-function in the Figure 2.5 After all the columns have been updated, the four diagonals are updated. In

Blake-256, a full or complete state update consists of 14 rounds of state update. A G-function operation on a state column or diagonal uses 2 message words. Thus, each round of state update utilizes 16 message words i.e. 512 bits message block [12].

2.12 Permutation Table

The permutation table is shown in Table 2.2. Permutation table σ[12]is used to identify the index value for message and constant given input to G-function for each round. Message block represented as an array of message words m(0) to m(15); similarly, the constants employed in the G-function are represented as an array of constant words c(0) to c(15). The notation mσr (2i) represents a message word whose array index is determined by the permutation σr (2i). For example, if the current round of the state update, round r = 2 and the second column of the state is being updated (i = 1) then σr (2i) = σ2; from permutation table this gives a value of 12. Thus, mσr (2i) = m (12). Similarly, for cσr (2i+1), σr (2i+1) = σ3; from permutation table this gives a value of 0. Thus, cσr (2i+1) = c (0). In this way, sigma values of each round can be found.

Table 2.2 Permutation Table[12]

σ_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ_1	14	10	4	8	9	15	13	6	1	12	0	2	11	7	5	3
σ_2	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
σ_3	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8
σ_4	9	0	5	7	2	4	10	15	14	1	11	12	6	8	3	13
σ_5	2	12	6	10	0	11	8	3	4	13	7	5	15	14	1	9
σ_6	12	5	1	15	14	13	4	10	0	7	6	3	9	2	8	11
σ_7	13	11	7	14	12	1	3	9	5	0	15	4	8	6	2	10
σ_8	6	15	14	9	11	3	0	8	12	2	13	7	1	4	10	5
σ_9	10	2	8	4	7	6	1	5	15	11	9	14	3	12	13	0
σ_{10}	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ_{11}	14	10	4	8	9	15	13	6	1	12	0	2	11	7	5	3
σ_{12}	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
σ_{13}	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8
σ_{14}	9	0	5	7	2	4	10	15	14	1	11	12	6	8	3	13
σ_{15}	2	12	6	10	0	11	8	3	4	13	7	5	15	14	1	9

2.13 G-Function

The architecture of G-function is given in Figure 2.6. The G-function has inputs of 4 state variables, 2 message words and 2 constant words. It has 4 outputs which are same 4 inputted state variables [10]. It mainly includes three operations: the modular adder of 2^n, the bit-by-bit XOR on n-bit words and right rotation operation of k-bits. The G-function performs operations such as addition, rotation and XOR on the state variables.

In the operations described in Figure 2.6 symbol " >>> " represents a rotation to the left and symbol " ^ " represents a bitwise XOR operation. a, b, c and d are state variables. From figure 3, the state column or diagonal denoted by the subscript i in the notation Gi is evident. The message words (m) and constants (c) utilized for G-function calculation are determined by a permutation table named σ.

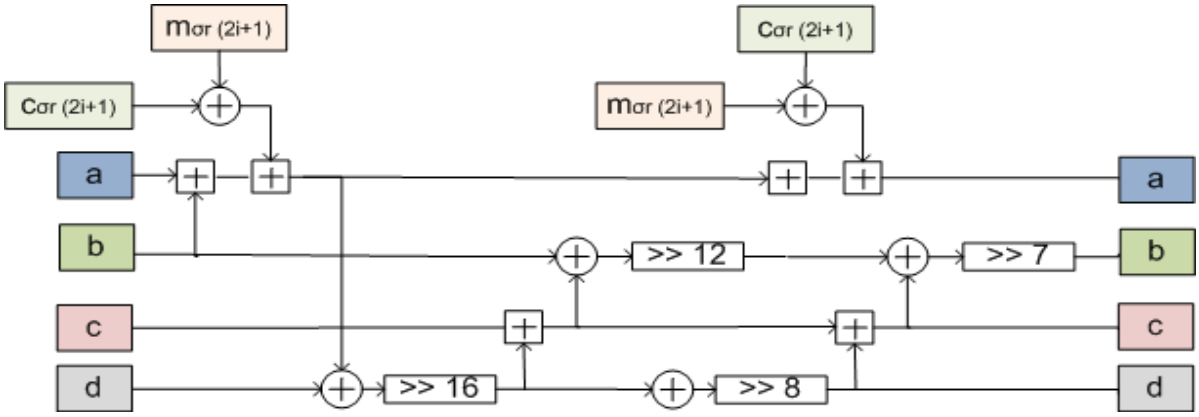

Figure 2.6G-Function Architecture[12]

The constants utilized in the G-function (such as (2i)) are determined in a similar manner. The array of 16 constants is given in Table 2.3 below:

Table 2.3Blake-256 Constants[12]

c0	243F6A88	c1	85A308D3
c2	13198A2E	c3	03707344
c4	A4093822	c5	299F31D0
c6	082EFA98	c7	EC4E6C89
c8	452821E6	c9	38D01377
c10	BE5466CF	c11	34E90C6C
c12	C0AC29B7	c13	C97C50DD
c14	3F84D5B5	c15	B5470917

Table 2.4 Blake-256 Initialization Value[12]

IV0	6A09E667F3BCC908	IV1	BB67AE8584CAA73B
IV2	3C6EF372FE94F82B	IV3	A54FF53A5F1D36F1
IV4	510E527FADE682D1	IV5	9B05688C2B3E6C1F
IV6	1F83D9ABFB41BD6B	IV7	5BE0CD19137E2179

2.14 Finalization

The finalization stage is the last process in the computation of the hash value of a message block. After the rounds execution, the new chain value $h' = h0', \ldots, h7'$ is extracted from the state $v0, \ldots, v15$ with input of the initial chain value $h = h0, \ldots, h7$ and the salt $s = s0, \ldots, s3$ as shown in Table 2.5. This hash value may be a chain value or the digest of the message.

Table 2.5 Finalization (Hash Value calculation)[12]

$h0'$	$h0 \oplus s0 \oplus v0 \oplus v8,$
$h1'$	$h1 \oplus s1 \oplus v1 \oplus v9,$
$h2'$	$h2 \oplus s2 \oplus v2 \oplus v10,$
$h3'$	$h3 \oplus s3 \oplus v3 \oplus v11$
$h4'$	$h4 \oplus s0 \oplus v4 \oplus v12,$
$h5'$	$h5 \oplus s1 \oplus v5 \oplus v13$
$h6'$	$h6 \oplus s2 \oplus v6 \oplus v14,$
$h7'$	$h7 \oplus s3 \oplus v7 \oplus v15$

When the salt input is not used, its value is set to 0 and it simply functions as a constant. Table2.4gives the initial values used for the first block of a message (referred to as the initialization vector) in both the state initialization and the finalization processes. For subsequent message blocks the initial value is given by the hash code (chain value) of the previous message block.

2.15 Summary

Detail specifications and complete architecture of Blake algorithm is discussed in this chapter along introduction of hash functions and NIST competition for SHA-3. Hash function is a transformation that takes a variable-size input m and returns a fixed-size string. Secure Hash Algorithm (SHA-2) family of hash functions developed by the National Institute of Standards and Technology (NIST) is currently used as standard for Hash Functions. Serious attacks have been reported in recent years against cryptographic hash algorithm, including SHA-1, and SHA-2 family, NIST issued a Call for a New Cryptographic Hash Algorithm (SHA-3). Five Secure Hash Algorithms were selected after Round-3 of SHA-3 competition. Blake is one of the candidates of round three of SHA-3 competition, based on Haifa iteration mode. Internal structure of Blake is the local wide-pipe and its compression algorithm is a modified version of Bernstein's stream cipher 'ChaCha'. Blake algorithm consists of family of four hash functions. The core of the design is based on compression function, which includes G-Function calculation. The compression function involves 8 instances of G-function. G function consists of addition, XOR and rotation operations. Blake algorithm is divided into three stages Initialization, Round Function and Finalization. After initialization the compression function iterates a series of 14 rounds. A round is a transformation of the state v that includes G-function execution on v state values firstly column wise and secondly in diagonal in each round. After, finalization process based on state registers results of round function, initial value and salt value, final hash value is calculated. In the next chapter, we will discuss the various design methodologies and optimization techniques of Blake algorithm implementation on FPGA used by various researchers.

CHAPTER 3: BACKGROUND AND LITERATURE REVIEW

3.1 Design Methodologies

Different design methodologies are used for the implementation of Blake algorithm on FPGA. These methodologies can the divided into two broad categories; High Speed and Low Area implementations. The division is based on the extent of serialization involved in their implementation. Those architectures used for high speed implementations, based on parallel design in which number of processes executed at a same time. While, in low area implementations, process is executed serially with same hardware in iterative mode. In full parallel design of Blake architecture, 8 G-Functions is implemented in parallel with a feedback registers. In serial design of Blake algorithm, 1 or part of G-Function is used with state resisters to stores intermediate state and execute in iterative mode to calculate overall hash value and4G design constitutes 4 G-Functions uses both the features of parallel and serial architecture. Brief description of optimization techniques used for the implementation of Blake algorithm is given below.

3.1.1 Folding

The widely used approach for the low area implementation of Blake algorithm is folding [14]. Data-path width is reduced in vertical folding, while horizontal folding reduces the size of processing elements while maintaining the width of data-path. Using this technique, Processing elements are reused and area resources are reduced, while numbers of clock cycles are increased.

3.1.2 Efficient use of Slice architecture

Each slice of Virtex 5 FPGA consists of four 6-input LUTs and a carry chain with a series of four multiplexers and four XOR gates, collectively known as CARRY4 primitive. Modulo-32 addition is performed during G-Function calculation. Exploiting 6-input LUT along with CARRY4 primitive gives fast implementation of modulo-32 addition. These LUTs can be used as single 6-input LUT or two 5-input LUTs with same inputs. Thus, selecting a logic such that large number of LUTS could be used as dual 5-input LUTS. This certainly reduces the slice count of the design. Also, use of intermediate registers and feedback registers are recommended

to implement using Distributed memory slice architecture because using LUT as DRAM reduces the delay path and hence, faster design implementation could be achieved [14].

3.1.3 Multi-Stage Pipelining& Rescheduling

Another optimization technique, mostly used for Blake algorithm optimization on FPGA is Pipelining and rescheduling of G-Function sequence. These design techniques are useful for low-area architecture. Round function of Blake algorithm is symmetric in nature, it can be split into various sub-processes and intermediate registers are added to introduce pipe-lines. This remarkably reduces the critical path but more clock cycles are required for round completion. Considering LUT-based FPGAs, additional registers do not require a large area, because these registers can be mapped together with the logic resources in the same slice. Also, pipelined computation of the complete compression function basically does not require additional clock cycles and the maximum operating clock frequency will be higher. Therefore, the throughput-area ratio increases.

Blake G-Function constitutes two similar halves, with a difference of only rotation constants. In this regard, most famous design is based on single Half-G Function where appropriate registers are added to introduce pipe-line stages. Instead of calculating G-Functions in a sequence, rescheduling is applied and G-functions are calculated in order such that after G0, G1, G2, G3 calculation, G7, G4, G5, G6 are calculated to avoid pipeline install. Feedback registers are required to store intermediate values of each half function calculation. In this way, design can be implemented with least hardware, least critical path and with efficient pipeline stage implementation overall TPA can be improved.

3.1.4 Using Dedicated FPGA Resources

Instead of using slice architecture, dedicated FPGA resources could be used for optimization like Block RAM, DSP Slices and Multipliers. Use of dedicated resources not only reduces the slice count but also increase the speed due to internal fast propagation of these hardware resources. Permutation Table used for sigma value calculation during each round can be stored in Block RAM of FPGA. Also, intermediate registers can be utilized using dedicated Block Memory. Internal G-Function of Blake algorithm is based on addition and multiplication operation, these

operations could be performed on dedicated Multiplier and DSP resources to overall reduce the slice count of FPGA.

3.2 Related Work

Various design techniques are used for both high-speed and compact design implementations for Blake algorithm as discussed in previous section. Kris et al. [15]implemented different architectures of Blake algorithm based on folding and pipelining technique. Detail analysis of Vertical and Horizontal Folding with different factor has been performed in his work. Horizontal folding with factor of 2 is similar to our 4G design, horizontal folding with the factor of 4 is similar to 1G and x1 basic iterative design is similar to our 8G design.

Baldwin et al. [16]design is based on older version of Blake algorithm submitted in second round of SHA-3 competition. In this version only 10 rounds are required for Hash value calculation. In his implementation, compression function is divided into two identical sections where round completion takes 4 cycles and counter, salt and message values are stored in BRAM of FPGA.

Kerchof et al. [17] used two implementation strategies; one with timing performance and other with area reduction. Fully autonomous architecture is implemented in his design where pipelining mechanism is used and rescheduling is performed to avoid pipeline stall. Baldwin et al. [16]and Kerchof et al. [17]used the similar approach as we used in our 1G design.

Benhard[18]used 2 half G-Functions in parallel with pipelining in his design. Rescheduling is applied to avoid pipeline stall. This design is comparable with our 1G design. Nikolas et al. [19]used a compression function where all message, state, constant; salt and count values are stored in Block RAM of FPGA. Kaps et al. [20] proposed two design architectures; one with block RAM and other one used logic resources. Instead of using full G function, one Half G-Function with Quasi pipelined stages are used in their design for compression function.

Kashif et al. [21] implemented both 256- and 512- variants of Blake algorithm on FPGA hardware. In his implementation, Blake algorithm is designed on Verilog HDL using LUT primitives of FPGA. In their design, 4 G-Functions were executed in parallel and each round is calculated in 2 clock cycles which is similar to our 4G design.Virtex 5, Virtex 6 and Virtex 7 FPGAs were used as hardware platform for implementation.

Beauchat et al. [22] used the approach of co-processor for the calculation of compression function of Blake algorithm. Pipelined architecture was used with interleaving of four compression functions. The design is based on Block RAM of FPGA. Both design implementations are based on Round-2 version of Blake algorithm. Miroslav et al. [23]design was also based on Round-2 of SHA-3 competition. In his fully autonomous implementation, analysis of IO interface on overall design performance is discussed.

Vaibhav et al. [24]used 4 G-Functions in their design.Blake-32 design is implemented on Virtex 5 FPGA which is based on round-2 submission of Blake algorithm.

Most of the design work for the implementation of Blake algorithm on FPGA was focused on Round Function design. Other important factors that affect overall implementation performance are also discussed in this work. These factors are message and constant value storage and selection mechanism, Finite state machine for controlling operation and efficient multiplexer implementation which greatly affects the area and timing results of the design. Lot of work has been done so far by proposing optimized architecture of Blake algorithm, but no such effort has been performed to evaluate various design architectures of Blake algorithm at the same time. Comprehensive comparison of different design techniques used by various contributors based on 8G, 4G and 1G design along with implementation results is given in Table 3.1.

For Blake-512, Table 3.1shows that Vaibhav et al. [24]gives most efficient results in terms of throughput when comparing 4G Designs. Highest throughput per area (TPA) is obtained by Kashif et al. [21].They use 4-G design technique. Lowest slice count is achieved by Kirchof et al. [17]by applying 1G design methodology.

For Blake-256, Aumasson et al. [12] gives most efficient results in terms of throughput when utilizing 8G Design. Highest throughput-per-Area (TPA) is obtained by Benhard et al. [18] for 1-G design technique. Also, Aumasson et al. [12] gives lowest slice count by applying 1G design methodology.

Table 3.1 Blake Implementation Results

References	Version	Slices	Frequency (MHz)	TP (Gbps)	TPA
	8G				
Kris et al. **[15]**	Blake-256	2306		2561	1.11
Aumasson et al. **[12]**	Blake-32	1694	67	3103	1.83
Kris et al. **[15]**	Blake-512	3984		3401	0.85
Aumasson et al. **[12]**	Blake-64	4329	35	2389	0.55
	4G				
Kris et al. **[15]**	Blake-256	1691		2253	1.33
Kashif et al. **[21]**	Blake-256	1382		2290	1.66
Vaibhav et al. **[24]**	Blake-32	1301	50	1280	0.98
Aumasson et al. **[12]**	Blake-32	1217	100	2438	2.00
Kris et al. **[15]**	Blake-512	3337		3159	0.95
Kashif et al. **[21]**	Blake-512	2582	100.02	3210	1.24
Vaibhav et al. **[24]**	Blake-64	11800	27	9804	0.83
Aumasson et al. **[12]**	Blake-64	2389	50	1766	0.74
	1G				
Kris et al. **[15]**	Blake-256	1547		1770	1.14
Baldwin et al. **[16]**	Blake-32	1118	118.1	1169	1.05
Benhard et al. **[18]**	Blake-256	374	163	725	1.94
Kirchof et al. **[17]**-area	Blake-32	192	240	183	0.95
Kirchof et al. **[17]**-speed	Blake-32	215	304	232	1.08
Aumasson et al. **[12]**	Blake-32	390	91	575	1.47
Kris et al. **[15]**	Blake-512	2935		2287	0.78
Baldwin et al. **[16]**	Blake-64	1718	90.9	1299	0.76
Aumasson et al. **[12]**	Blake-64	939	59	533	0.57

3.3 Summary

In this chapter, we have discussed various design techniques and methodologies used for Blake implementation on FPGA. There are mainly three types of architectures use for implementation of cryptographic Hash Function on FPGA. We have implemented fully autonomous design in

which all design features are implemented on slice resources of FPGA. Fair comparison with other contribution is assured by using similar implementation technology and platform used by various researchers. The performance metric to analyze the efficiency of the design is based on maximum design frequency, maximum throughput and number of slice count of FPGA. Different design methodologies used for the implementation of Blake algorithm on FPGA. These methodologies constitute High speed and Low Area implementations. The division is based on the extent of serialization involved in their implementation. Other optimization techniques includes Folding data path and process, efficient use of slice resources, multi-stage pipelining and rescheduling and using dedicated FPGA resources. Lot of efforts based on these design techniques are performed by various researchers for efficient design implementation of Blake algorithm on FPGA. Next chapter will describe the detail of our design methodology for implementation of Blake algorithm.

CHAPTER 4: DESIGN APPROACH

4.1 BLAKE Implementation on FPGA

In our design we have implemented Full Autonomous architecture of Blake algorithm which include initial function, round function and finalization where all design logic is implemented on Slice architecture of FPGA. All of the intermediate values in registers during the hashing process are stored in Distributed Memory (DRAM) of FPGA and no external memory or Block RAM (BRAM) is utilized.

Considering the different design methodologies for the implementation of Blake algorithm on FPGA, as discussed in previous chapter, we have implemented three design architectures of Blake algorithm named as 8G, 4G and 1G in this research work and optimization is performed each design architecture. These architectures are based on extent of serialization of different processes performed in Round Function calculation of Blake algorithm. Initialization and Finalization process is same for all these schemes and optimization of Round Function is the core objective of our design. Round Function of Blake Algorithm is based on G-Functions as discussed in chapter 2. In the first design we have used 8 G-Functions in parallel, second design used hardware of 4 G-Functions and the third design is based on 2 half-G Functions in parallel with pipeline registers between them.

4.2 Performance Evaluation

In addition to security, performance is one of the main criteria for evaluating the SHA-3 candidates [7]. To evaluate the hardware efficiency of different architectures of Blake algorithm, we have fixed certain features of design implementation. Since, different designers used different hardware platforms, synthesis and implementation tools and different I/O interfaces, fair comparison is difficult to perform. We have used the features which are mostly used in research contributions andour design used only Slice resources of FPGA. Virtex 5 is used in most of the evaluation work done so far for the implementation of Blake algorithm. Therefore, we have used same device family in our implementation.

4.3 Performance Metrics

To compare different architectural designs of Blake algorithm, metrics are required for the evaluation of design architecture. The evaluation is characterized by the following three primary metrics.

Area: The area of a design on an FPGA is calculated in terms of the number of logic cells it occupies. Basic logic cell of FPGA architecture is known as slice. Area metrics indicates of the resource utilization of a design calculated in term of number of slices used.

Maximum Frequency: The timing information is extracted from the Xilinx TRACE reports generated after Place and Route operation. It indicates the design capacity at what maximum frequency the design can operate without any fault. This is based on the timing of critical path delay and measured usually in MHz.

Maximum Throughput: The throughput of a design is the measure of the maximum data rate of the design at the output of the core. It is calculated using maximum frequency results generated after place and route operation. The formula for calculating maximum throughput is given in Equation 4.1 where the block size is the amount of data the hashing algorithm will process at a time.

$$Throughput = Input\ Block\ Size\ \times \frac{Max\ Clock\ Frequency}{Number\ of\ Clock\ Cycles} \qquad [4.1]$$

For Blake-256 it is of size 512 bits and for Blake-512 it is 1024 bits. As Hash functions have iterative nature, the number of clock cycle indicates the total time required to calculate the complete Hash value. Apart from these performance metrics, other factors like Power and Energy are also used as performance metrics of the algorithm. But, here in this work, we will focus only on first three parameters as these are enough for fair comparison of the different architectures and with other research contributions.

4.4 Common Design Features

Implementation of Blake algorithm on FPGA constitutes Initial Function, Round Function and Finalization process. Wrapper Module is required to provide input and output interfaces of Blake

algorithm and Finite state machine controller is performed all the control operation of the design. Round function calculation require sigma values generated through message and constant select mechanism from the input array of message and constant values for each round. Wrapper design, finite state machine controller and strategy for the selection of input message and constant values are similar for all three architectures and their description is given below, while the detail descriptions of each architecture is given in subsequent sections.

4.4.1 Wrapper Design

We cannot directly implement Blake Algorithm design on Virtex 5 FPGA due limited IO resources. It input message size for Blake-256 will be 512-bits and output hash value is 256-bits, which cannot be directly mapped to FPGA IOs. Similarly, for Blake-512, input message size will be 1024-bits and output hash value is 512-bits. Therefore, a wrapper is required to implement the algorithm on FPGA. Wrapper includes 64-bits input interface while 256-bits hash value is directly connected to the output port of FPGA. The wrapper consists of input FIFO as shown in Figure4.1, while padding function is considered to be performed outside the wrapper design.

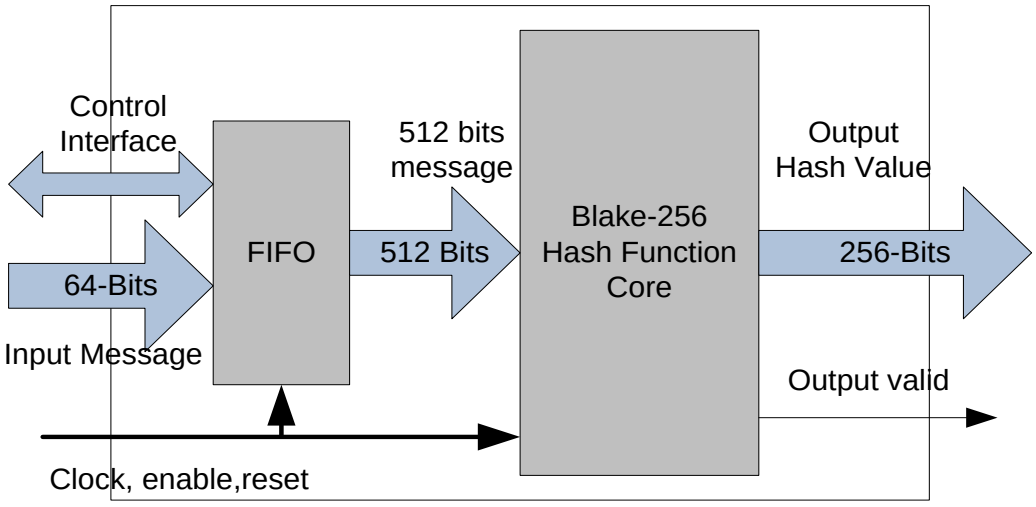

Figure 4.1 Blake-256 wrapper architecture

4.4.2 Message/Constant Select Mechanism

Blake algorithm requires permutation table to calculate the index value of message and round constants. Permutation table is implemented on distributed memory of FPGA and counter is used for memory addressing. Multiplexers are used for message and constant value selection. The

sigma value calculated for each round is buffered to reduce the critical path. The general structure of message and constant value selection mechanism for each design is shown in Figure 4.2.

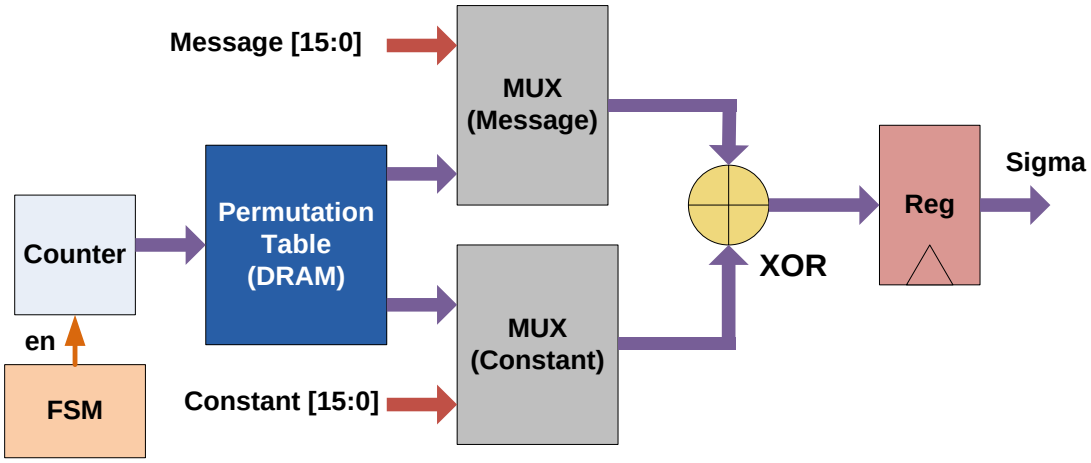

Figure 4.2 Message and Constant Select Mechanism

4.4.3 FSM Controller

Finite state machine is used to control the overall operation of Blake algorithm. Basic function of state machine is to control round counter and controlling of counter for permutation table indexing on DRAM. ROM-based FSMs are more efficient in terms of area consumption and speed compared to conventional FSM based on BRAM implementation and their maximum frequency is independent of the complexity [9]. The area required to implement ROM-based FSMs is determined by the number of control signals and states. We have used Moore state machine with one-hot encoding for state register. Minimum state approach is used to reduce the hardware cost.

4.5 8G Design

4.5.1 Round Function

The architecture of 8G design is given in Figure 4.3. It is consist of 8 G-Functions. 4x4 initial vector given to the round function through input Multiplexer in first cycle. The output generated in first cycle will be stored in a 4x4 32-bits register matrix. The register matrix values will be given back to input through input multiplexer. Since, all G-Functions are connected in parallel;

each round will be executed in one cycle. Sigma values are calculated from message and constant values for each round through mechanism given in Figure 4.4.

The major advantage of 8G architecture is only one cycle is required for one round and whole algorithm is computed in 14 clock cycles for Blake-256. No additional multiplexers are required for order variation between column and diagonal step and signals are directly connected as shown in Figure 4.1 and also no additional registers are required between column and diagonal step. Major disadvantage of this design is largest path delay and hence, it is difficult for synthesis and implementation tool to optimize such a large data path.

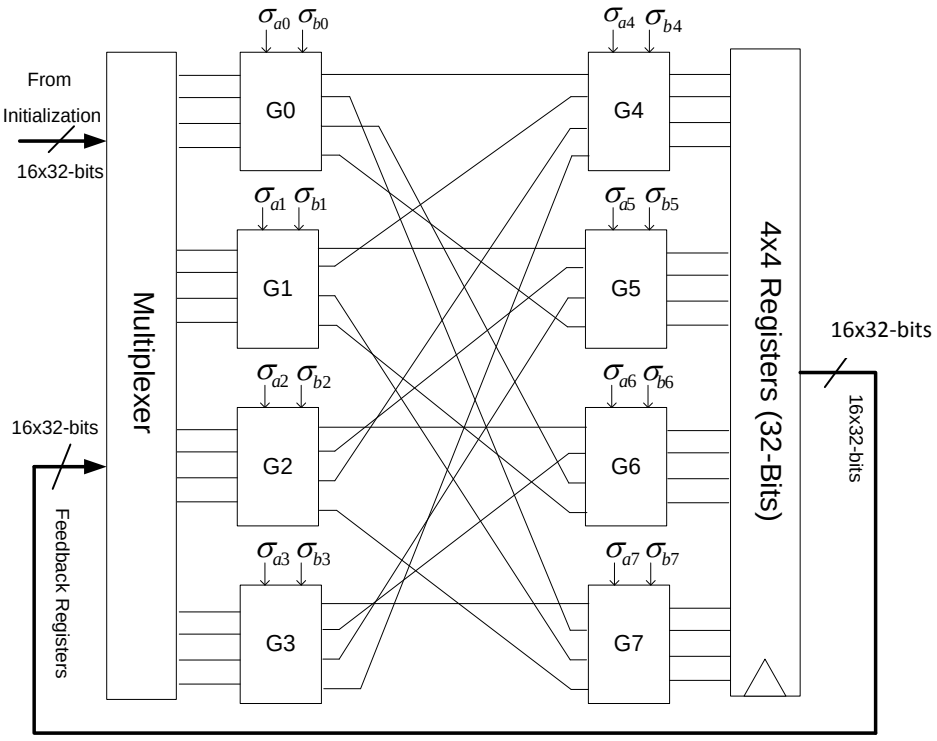

Figure 4.3 8G Design Architecture

4.5.2 Message/Select Mechanism

Since, 8 G-Functions are executed at a time, sigma a0, sigma b0,.., sigma7,.., sigmab7 values required as shown in Figure 4.3, will be generated using counter, distributed memory, multiplexer and xor gates as shown in Figure 4.4. The permutation table is stored in distributed memory of FPGA.

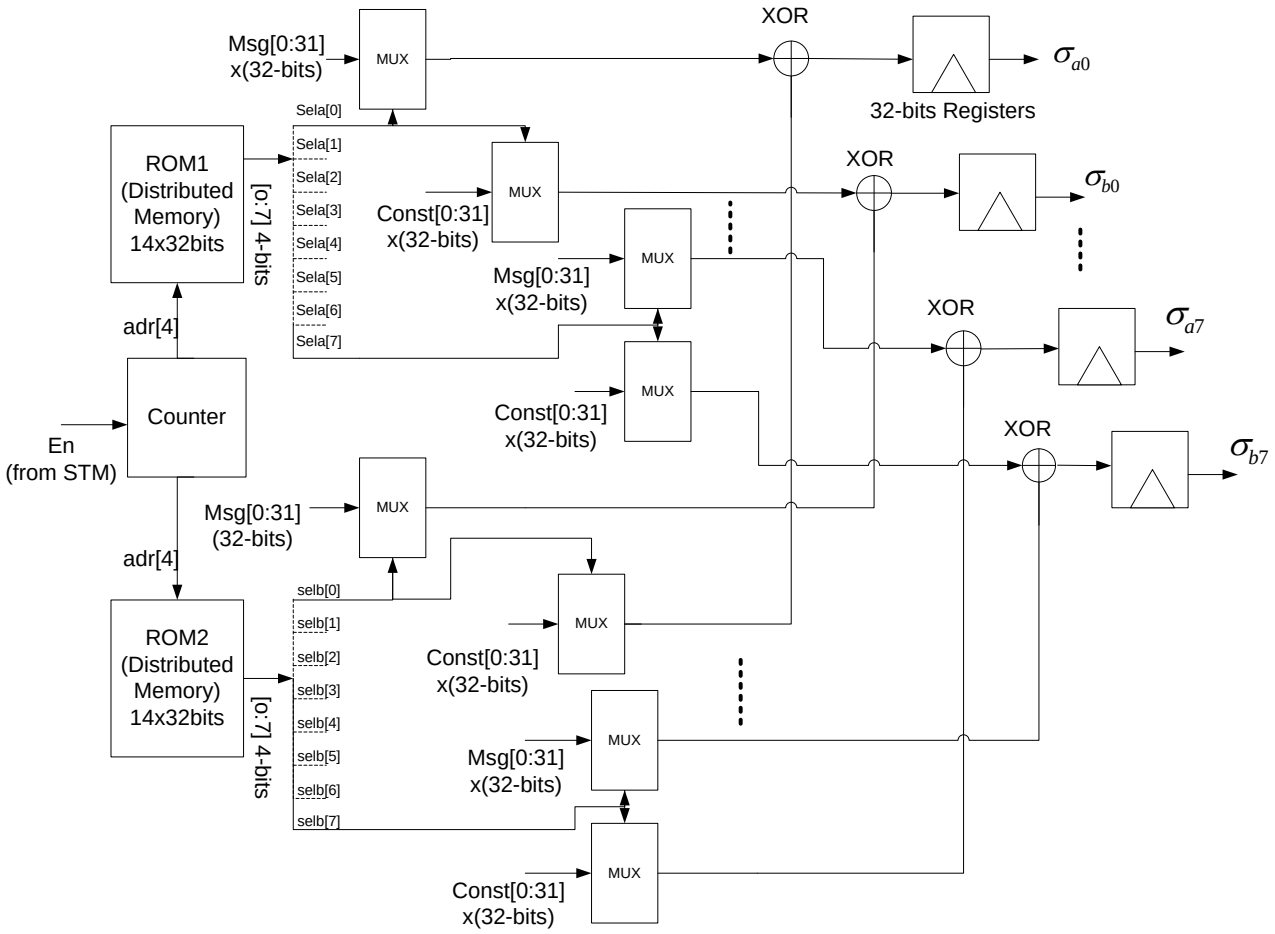

Figure 4.4 8G Design message and constant selection

The memory will act like ROM and two ROM cores are generated having sizes of 14x32bits each. 3216x1 Multiplexers are used for message and constant value selection and Xoring operation is performed according to the Equations 4.2 and 4.3. Registers are used to separate the critical path of round function with this DRAM chain.

Sigma ai = $c\sigma r2i{+}1 \; + m\sigma r2i$ [4.2]

Sigma bi = $c\sigma r2i + m\sigma r2i{+}1$ [4.3]

4.6 4G Design

4.6.1 Round Function

The architecture of 4G design is given in Figure 4.5. It is consist of 4 G-Functions and 4x4 initial vectors given to the round function through input Multiplexer in first cycle. The column step

output generated in first cycle will be stored in a 4x4 32-bits register matrix. The register matrix values will be given back to input through input multiplexer and signal router will re-order the state values according to the diagonal step sequence. Since, one cycle is required for column step execution and one cycle is required for diagonal step execution; each round will be calculated in two clock cycles. Sigma values calculated from message and constant values for each round through mechanism given in Figure 4.4. For column step calculation the signal router will route the signal directly in straight order, while for diagonal step order will be change depending upon the ‘col_diag_sel’ signal value i.e. ‘0’ for column step and ‘1’ for diagonal step.

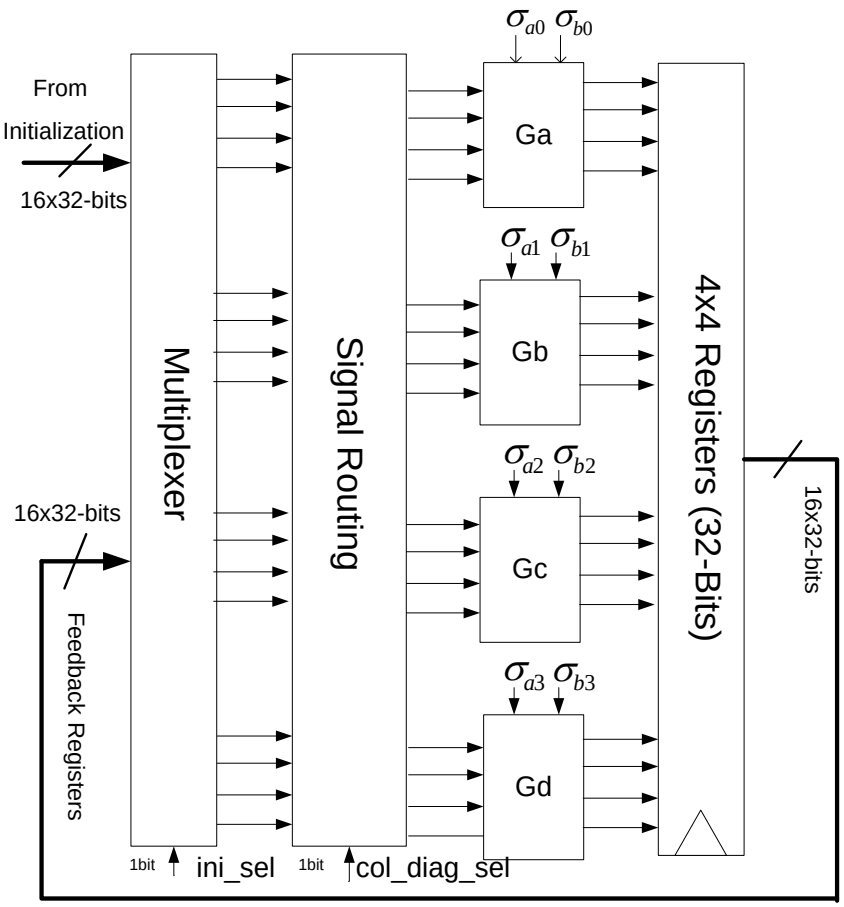

Figure 4.5 4G Design Architecture

The major advantage of this architecture is reduction in critical path to half as compared to 8G architecture. Two cycles are required for one round thus whole algorithm is computed in 28 clock cycles. Since, 8 sigma values are required for one cycle thus reduces the number of multiplexers required for message and constant values selection as given in Figure 4.3. Additional resources will be required for signal routing and slightly complex state machine will

be required for its control. Overall design methodology best suited the Virtex 5 architecture and hence, best results obtained in terms of TPA.

4.6.2 Message/Select Select Mechanism

For 4G architecture, 4 G-Functions are used, only column or diagonal step will executed at a time, thus sigma a0, sigma b0, .., sigma3, sigmab3 values required as given in Figure 4.3.These values are generated using the same methodology described for 8G architecture but only one ROM instance of size 28x32bits is used and 16 multiplexers are used for message and constant values selection as shown in Figure 4.4. Xoring operation is performed according to the Equations 4.4 and 4.5.

Sigma ai = $c\sigma r2i{+}1 \; + m\sigma r2i$ [4.4]

Sigma bi = $c\sigma r2i + m\sigma r2i{+}1$ [4.5]

4.7 1G Design

4.7.1 Round Function

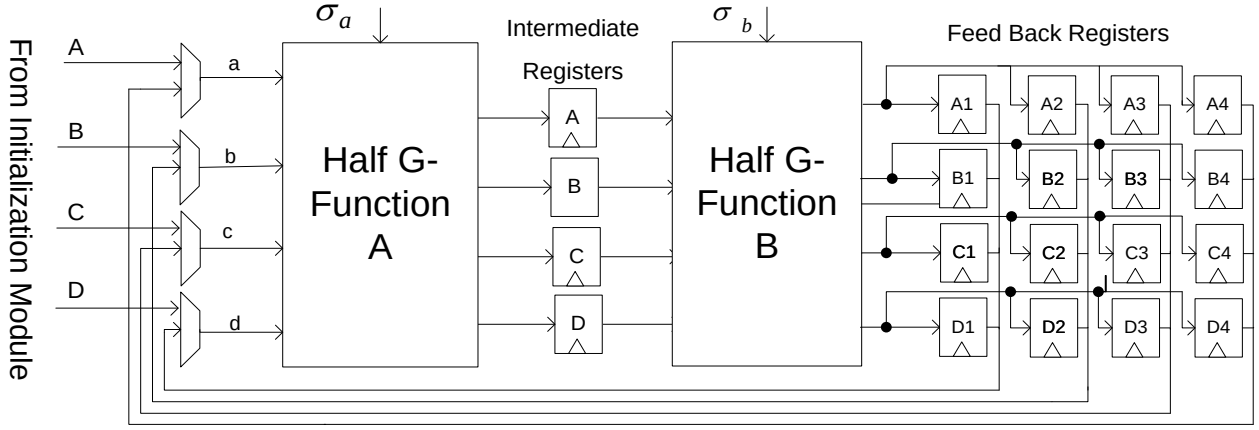

Figure 4.6 1G Architecture Design

The architecture of 1G design is shown in Figure 4.6. It includes two half-G Functions, four input multiplexers, four intermediate registers and 4x4 feedback register array. Input multiplexers will be responsible for selection of inputs for Half-G Function A. At first round, column step, initial state vector will be given to its input through input multiplexer. In diagonal step of first round input to the Half-G Function 'A' is given through feedback register array.

Intermediate registers will be act as pipeline registers, thus, two clock cycles will be require for one G-function calculation.

In first cycle first part of G-Function i.e. G0a is calculated and filled up the pipelined register as shown in Figure 4.7. In the next clock cycle, remaining part of G0 i.e. G0b will be calculated at the same time the G1a will be calculated too. In this way after five clock cycles, column step of first round has completed and its output will be stored in 4x4 feedback register.

For diagonal step of round 1, inputs to Half-G Function A is selected through input multiplexer from 4x4 feedback register matrix in a sequence required by diagonal step. Each output of Half-G Function B is connected with 4 Registers simultaneously. The 'en' signal of the register will determine that selection of the output register to store intermediate state value. The critical path of the design consist of input multiplexer and five operations between points 'a' to 'b' of Half-G Function represented in Figure 4.6. The Xoring operation of message and count values will not be included in critical path as the two registers have been used for σ_a and σ_b as mentioned in Figure 4.8. Therefore, total 10 clock cycles are required for each round calculation. Thus, 10*14 = 140 clock cycles are required for complete hash value calculation for Blake-256 and 10*16 = 160 clock cycles are required for complete hash value calculation for Blake-512.

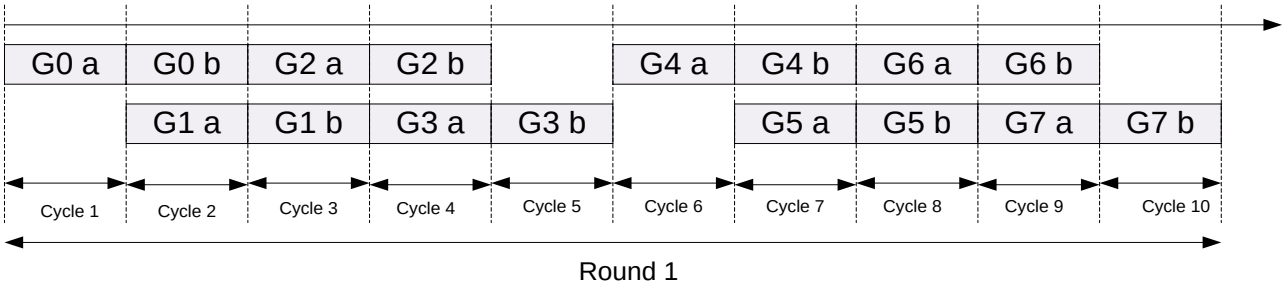

Figure 4.7 1G Design Flow (1 Round)

4.7.2 Message/Select Mechanism

The mechanism for message and counter values selection is shown in Figure 4.8 and similar to the 8G and 4G designs. Two ROM primitives with the size of (8x14) x 4-bits i.e. 56 bytes have been used and 7-bit Counter is used for address generation of two ROMs and it will be controlled

by FSM controller. The selection for message and count values for each round can be represented as given in Equation 4.6 and 4.7.

$$For\ HalfG\ Fn\ A\text{:}\quad constant\text{:}c\sigma r2i{+}1\ \ ;message\text{:}\ m\sigma r2i \qquad [4.6]$$

$$For\ HalfG\ Fn\ B\text{:}\quad constant\text{:}c\sigma r2i\ \ \ \ ;message\text{:}\ m\sigma r2i{+}1 \qquad [4.7]$$

It can be seen from above equations that same permutation index value for Half-G Function B will be given to Half-G Function B in next clock cycle. Hence, we have used registered/delayed ROM output to the multiplexer for message b and constant b as shown in Figure 4.8.

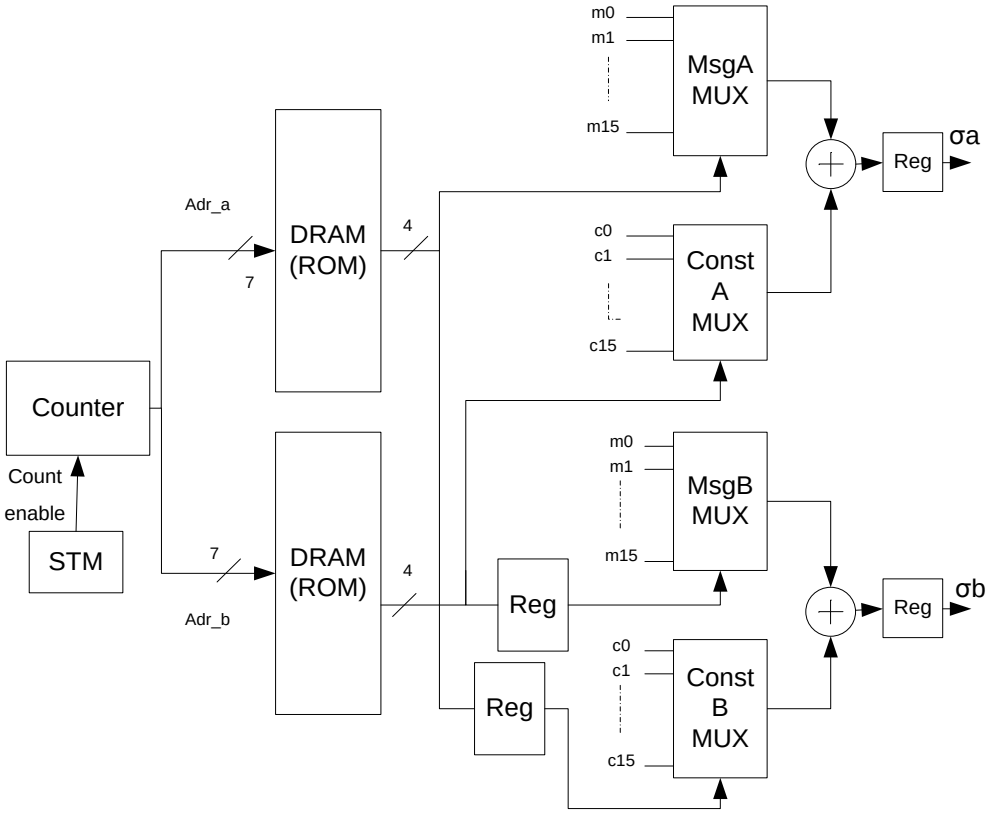

Figure 4.8 1G Design message and constant selection

4.8 Summary

In this chapter detail design approach for three proposed architecture of Blake algorithm. In other to evaluate the hardware efficiency of different architectures, certain features of design implementation have been fixed like I/O interface, FSM controller, wrapper, initialization

module and finalization modules. Three proposed designs include 8G which is consisting of 8 G-Functions. Only one cycle is required for one round. Major disadvantage is largest path delay and hence hard to optimize by the tool. Second one is 4G-Design. It is consist of 4 G-Functions. Each round is calculated in two clock cycles. It has half critical path as compared to 8G architecture. Additional resources will be required for signal routing and slightly complex state machine will be required but overall efficiency is best in terms of TPA. Last one, 1G-Design architecture, includes two half-G Functions, four input multiplexers, four intermediate registers and 4x4 feedback register array. 10 clock cycles are required for each round calculation. It gives best results in terms of hardware utilization. Therefore, 8G design best suited for High throughput application, 1G design for least hardware and 4G gives best results in terms of TPA. In the next chapter, implementation results of these architectures are given with the performance analysis in terms of area, speed and TPA.

CHAPTER 5: IMPLEMENTATION AND RESULTS

5.1 Optimization Strategies

Synthesis and implementation of the design is performed on Xilinx ISE 13.1. There are different design optimization techniques built-in in ISE software based on various Synthesis, Place, Map and Route properties. Applying one strategy gives very different results from other. These strategies mainly based on “Speed”, “Area” and “Balance” optimization. Three different design strategies selected in ISE software are given in Table4.1.

Table 5.1Design optimization strategies

Speed	Timing Performance without IOB packing
Area	Area Reduction with Physical Synthesis
Balance	Xilinx Default

5.2 Post Place and Route Implementation Results

Three design architectures 8G, 4G and 1G have been implemented using these three optimization strategies discussed above one by one. The post-route implementation results for Blake-256 are given in Table 1. Implementation results shows that maximum throughput of 2.62 Gbps is obtained from 8G design using “Speed Optimization” strategy. 4G design implementation with “Speed Optimization strategy” gives maximum TPA of 2.1. 1G design is best suited for low-area design implementation that gives slice count of 412 when using “Area Optimization” strategy as highlighted in Table 5.2.

The post-route implementation results for Blake-512 are given in Table 5.3. Implementation results shows that maximum throughput of 4.78 Gbps is obtained from 8G design using “Speed Optimization” strategy. 4G design implementation with “Area Optimization strategy” gives maximum TPA of 1.90. 1G design is best suited for low-area design implementation that gives slice count of 814 when using “Area Optimization” strategy as highlighted in Table 5.3.

Table 5.2 BLAKE-256 Post place and route implementation results on Virtex 5 FPGA

Strategy	Slices	Frequency (MHz)	TP (Gbps)	TPA
	8G			
Area	1450	61.767	2.259	1.558
Speed	2275	71.633	2.620	1.152
Balance	2653	63.816	2.334	0.880
	4G			
Area	929	98.232	1.796	1.934
Speed	900	105.263	1.925	2.139
Balance	1429	112.740	2.062	1.443
	1G			
Area	412	157.480	0.576	1.398
Speed	416	193.424	0.707	1.700
Balance	569	173.913	0.636	1.118

Table 5.3 BLAKE-512 Post place and route implementation results on Virtex 5 FPGA

Strategy	Slices	Frequency (MHz)	TP (Gbps)	TPA
	8G			
Area	2602	54.318	3.476	1.336
Speed	3206	74.794	4.787	1.493
Balance	4556	52.632	3.368	0.739
	4G			
Area	1478	87.951	2.814	1.904
Speed	1993	100.503	3.216	1.614
Balance	2172	99.900	3.197	1.472
	1G			
Area	814	142.857	0.914	1.123
Speed	801	161.290	1.032	1.289
Balance	1057	149.254	0.9555	0.904

The comparisons for Blake-256 implementation with other contributions are given in Table 5.4.

Table 5.4 BLAKE-256 Place and route results comparison (VIRTEX 5)

References	Version	Slices	Frequency (MHz)	TP (Gbps)	TPA
	8G				
Kris et al. **[15]**	Blake-256	2306		2.561	1.11
Aumasson et al. **[12]**	Blake-32	1694	67	3.103	1.83
This Work	**Blake-256**	**1450**	**61.76**	**2.258**	**1.56**
	4G				
Kris et al. **[15]**	Blake-256	1691		2.253	1.33
Kashif et al. **[21]**	Blake-256	1382		2.290	1.66
Vaibhav et al. **[24]**	Blake-32	1301	50	1.280	0.98
Aumasson et al. **[12]**	Blake-32	1217	100	2.438	2.00
This Work	**Blake-256**	**900**	**105.3**	**1.925**	**2.14**
	1G				
Kris et al. **[15]**	Blake-256	1547		1.770	1.14
Baldwin et al. **[16]**	Blake-32	1118	118.1	1.169	1.05
Benhard et al. **[18]**	Blake-256	374	163	0.725	1.94
Kirchof et al. **[17]**-area	Blake-32	192	240	0.183	0.95
Kirchof et al. **[17]**-speed	Blake-32	215	304	0.232	1.08
Aumasson et al. **[12]**	Blake-32	390	91	0.575	1.47
This Work	**Blake-256**	**416**	**193.4**	**0.707**	**1.70**

The Table5.4 shows that our design results have much better improvements as compare to other contributions. For 8G-Design, our design shows the TPA of 1.56 while Aumasson et al. [12] gives the TPA of 1.83. But, later one is based on Blake-32 which is older version and requires only 10 clock cycles for round completion. If TPA is calculated after considering 14 clock cycles for Aumasson, it will give TPA of 1.4 which is less than our design results. For 4G Design, our design result shows best performance for lowest slice count and highest TPA. 1G design results gives better performance as compare to all other design except Benhard. et al. [18], who used multiple pipeline approach with G-Function reorganization.

The comparisons for Blake-512 implementation with other contributions are given in Table 2. Our design results show much better improvements as compare to other contributions.

Table 5.5 BLAKE-512 Place and route results comparison (VIRTEX 5)

References	Version	Slices	Frequency (MHz)	TP (Gbps)	TPA
	8G				
Kris et al. **[15]**	Blake-512	3984		3.401	0.85
Aumasson et al. **[12]**	Blake-64	4329	35	2.389	0.55
This Work	**Blake-512**	**3206**	**74.79**	**4.787**	**1.49**
	4G				
Kris et al. **[15]**	Blake-512	3337		3.159	0.95
Kashif et al. **[21]**	Blake-512	2582	100.02	3.210	1.24
Vaibhav et al. **[24]**	Blake-64	11800	27	9.804	0.83
Aumasson et al. **[12]**	Blake-64	2389	50	1.766	0.74
This Work	**Blake-512**	**1478**	**87.9**	**2.814**	**1.90**
	1G				
Kris et al. **[15]**	Blake-512	2935		2.287	0.78
Baldwin et al. **[16]**	Blake-64	1718	90.9	1.299	0.76
Aumasson et al. **[12]**	Blake-64	939	59	0.533	0.57
This Work	**Blake-512**	**801**	**161.29**	**1.032**	**1.29**

5.3 Timing Comparison

Timing comparison of Blake-256 and Blake-512 design based on Throughput calculated using formula of equation is given in Figure 5.1. The comparison shows that 8G design gives highest throughput for both Blake-512 and Blake-256. In comparison of Blake-512 and Blake-256, Blake-512 gives higher throughput than Blake-256. Least throughput is resulted from 1G design.

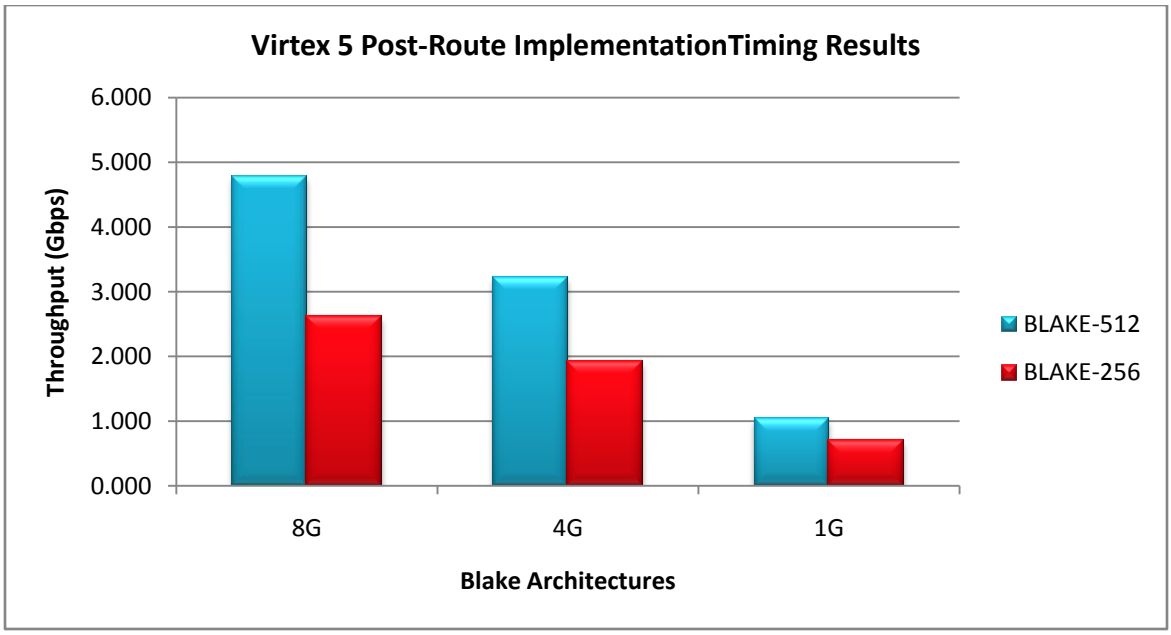

Figure 5.1 Timing Comparison

5.4 Area Utilization Comparison

Comparison of Blake-256 and Blake-512 area resource utilization based on number of slices used in the design is given in Figure 5.2. The comparison shows that 1G design has maximum efficiency in terms of resource utilization for both Blake-512 and Blake-256. As obvious due to larger data width, in comparison of Blake-512 and Blake-256, Blake-512 uses less number of FPGA slices. 8G design consumes highest number of slices.

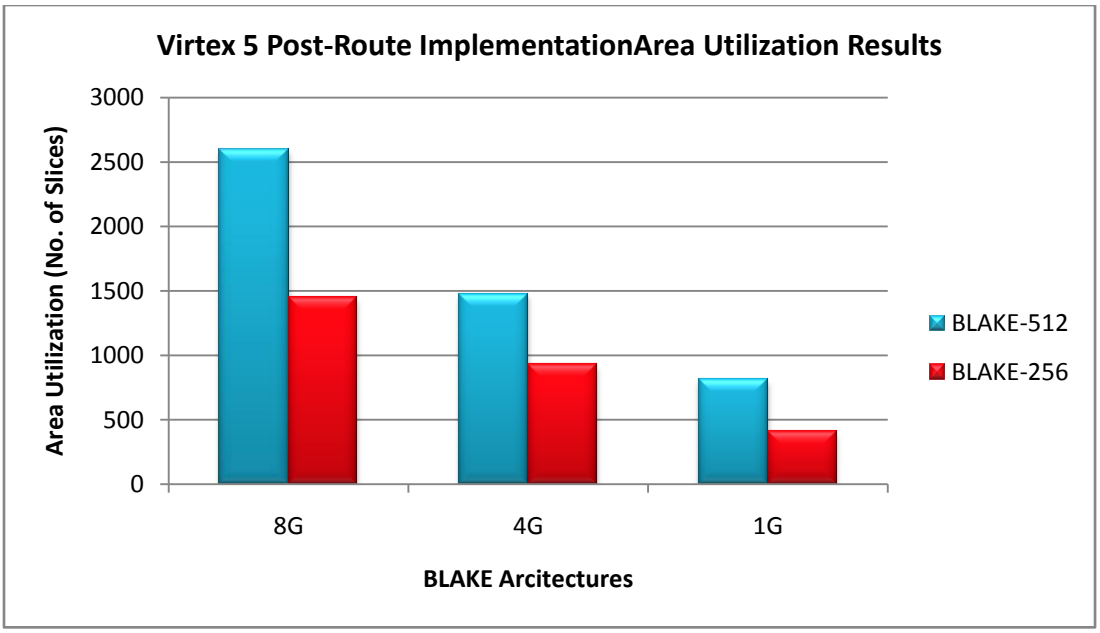

Figure 5.2 Resource Utilization Comparison

5.5 Throughput per Area

Usually, the efficiency of the Blake algorithm design on FPGA is measured in terms of Throughput-per-Area. Throughput-per-Area (TPA) results of Blake-256 for three architectures with three optimization techniques are given in Figure 5.3. The results show that 4G design is most efficient in terms of TPA. Highest throughput of 2.1Gbps is obtained for 8G design with speed optimization strategy selected. Design results of 8G with respect to optimization strategies as compare to 4G and 1G are unusual because of large data path of 8G consists of 8G functions in a serial path. The critical path delay of 4G design is consist of 4 G-functions and a multiplexer is the basis for most efficient design on Virtex 5 architecture.

Throughput-per-Area (TPA) results of Blake-512 for three architectures with three optimization techniques are given in Figure 5.4. The results show that 4G design is most efficient in terms of TPA. Highest throughput of 1.9Gbps is obtained for 8G design with speed optimization strategy selected.

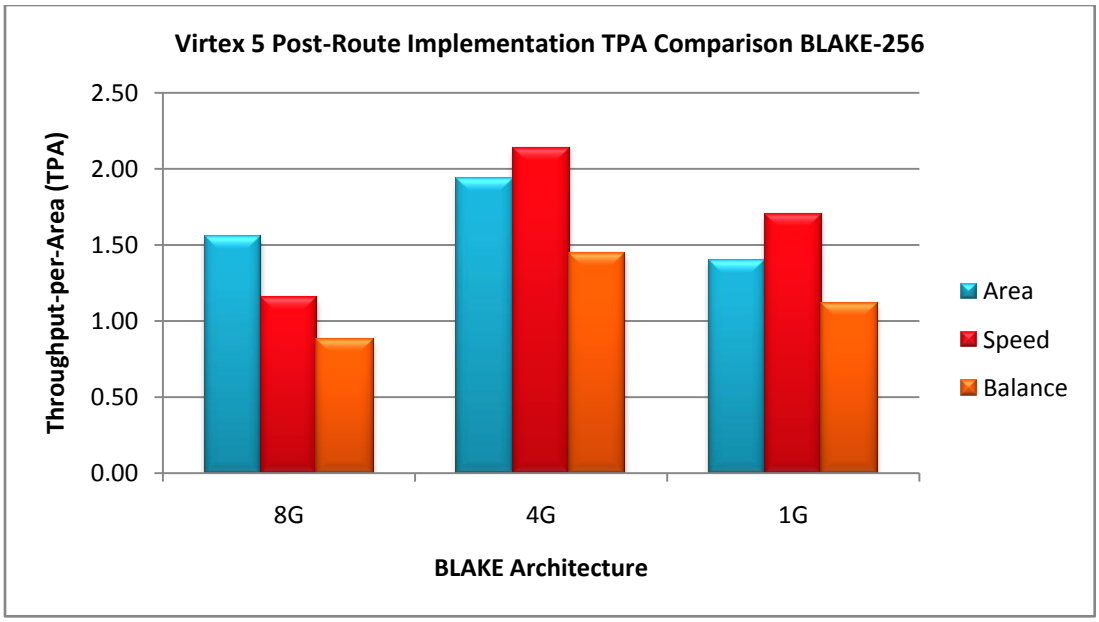

Figure 5.3 Throughput/Area Comparison for Blake-256

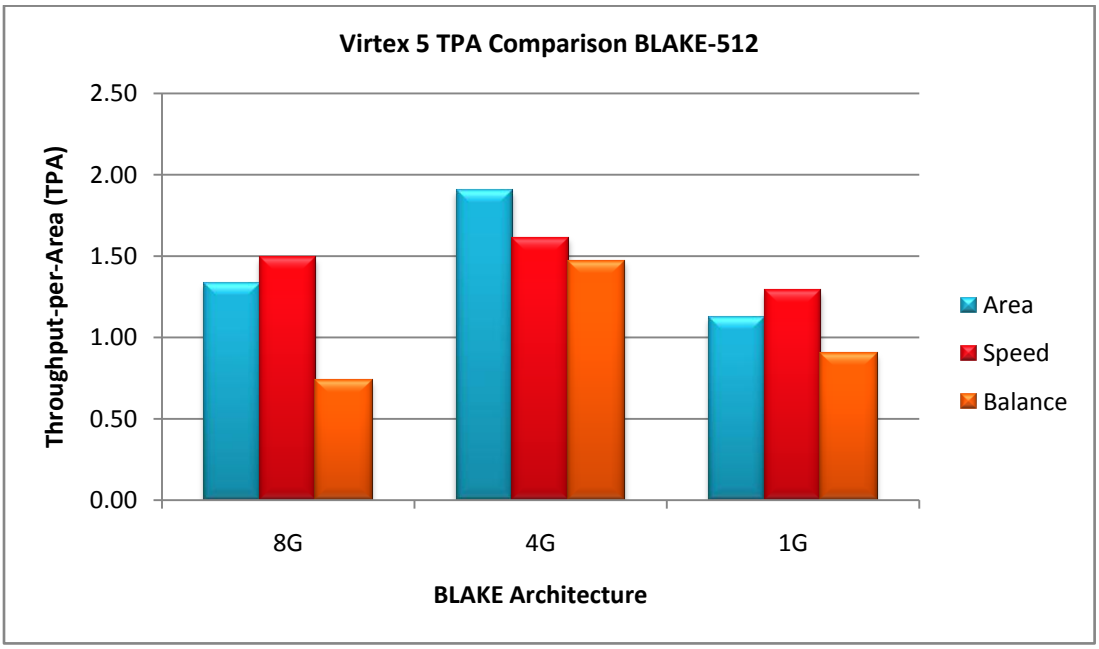

Figure 5.4 Throughput/Area Comparison for Blake-512

5.6 Summary

Post Place-and-Route Implementation Results are described in this chapter. Synthesis and implementation of the design is performed on Xilinx ISE 13.1. Three different design optimization techniques built-in in ISE software based on “Speed”, “Area” and “Balance” optimization are used for implementation. Three design architectures 8G, 4G and 1G have been implemented using these three optimization strategies. For Blake-256, implementation results shows that maximum throughput of 2.62 Gbps is obtained from 8G design using “Speed Optimization” strategy. 4G design implementation with “Speed Optimization strategy” gives maximum TPA of 2.1. 1G design is best suited for low-area design implementation that gives slice count of 412 when using “Area Optimization” strategy. For Blake-512, throughput of 4.78 Gbps is obtained from 8G design using “Speed Optimization” strategy. 4G design implementation with “Area Optimization strategy” gives maximum TPA of 1.90. 1G design is best suited for low-area design implementation that gives slice count of 814 when using “Area Optimization”. In the next and last chapter, concluding remarks along with some future recommendation of this research work are given.

CHAPTER 6: CONCLUSION AND FUTURE RECOMMENDATIONS

6.1 Conclusion

Different design architectures of Blake-256and Blake-512 are implemented on FPGA. Design uses large hardware resources gives maximum throughput i.e. 8G design requires only 14 clock cycles for Blake-256 and 16 cycles for Blake-512 for Hash value calculation resulted throughput of 2.6 Gbps and 4.7 Gbps respectively. Design uses least hardware resources gives best results in terms of area. 1G design gives most efficient results in terms of number of slices utilized. The optimized delay path is utilized in 4G design with respect to Virtex 5 architecture. That's gives maximum TPA of 2.1 for Blake-256 and TPA of 1.9 for Blake-512. Overall research suggests that selection of architecture is dependent upon type of application either high speed requirements or low area constraints, suitable optimization could be performed in a particular domain to achieve best design results.

6.2 Future Recommendations

In this research work, we have implemented both low area and high speed designs of Blake-256 and Blake-512. Our results show much better improvements as compare to previous contributions. The performance can be improved more by utilizing one of the dedicated resources such as DSP slices and Multipliers, internal G-Function operations. This will be helpful for low area implementation. Another optimization can be performed by increasing the pipeline stages in between G-Function to improve the timing performance. No work for the optimization of the design with respect to Power consumption is performed in this work. Power Analysis of proposed architectures could be performed in future, which is indeed an important factor in Hardware design evaluation. Implementation of proposed design is performed on Xilinx Virtex 5 FPGA, evaluation of the design on latest FPGA device like Virtex 6 and Virtex 7 and other devices families like Altera and Actel FPGAs could be performed in future for more comprehensive analysis of the design performance.

6.3 Publications

- Muhammad Arsalan and Dr. Arshad Aziz, "Compact Hardware Implementation of SHA-3 Finalist Blake on FPGA", 10th International Bhurban Conference on Applied Sciences & Technology (IBCAST), January 2013. [Accepted]
- Muhammad Arsalan and Dr. Arshad Aziz, "Comparative Analysis of high speed and low area architectures of Blake SHA-3 candidate on FPGA", 10th International Conference on Frontiers of Information Technology (FIT' 12), December 2012 [Published]

2 Bibliography

[1] M. Ilya, "Hash functions: Theory, attacks, and applications," Microsoft Research, Silicon Valley Campus, 2005.

[2] G. C. Kessler, "An Overview of Cryptography," 17 July 2012. [Online]. Available: http://www.garykessler.net/library/crypto.html.

[3] NIST, "Secured Hashing – Approved algorithms," 2011. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/secure_hashing.html.

[4] X. Wang, Y. Yin and H. Yu., "Finding Collisions in the Full SHA-1, Advances in Cryptology," in *CRYPTO 2005: 25th Annual International Cryptology Conference*, Santa Barbara, 2005.

[5] Yin, S. Michael and L. Yiqun, "Collision-Resistant usage of MD5 and SHA-1 via Message Preprocessing," in *Proceedings of the 2006 The Cryptographers' Track at the RSA conference on Topics in Cryptology*, Heidelberg, 2006.

[6] NIST, "Federal Register Vol. 72," November 2007. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf.

[7] NIST, "Cryptographic Hash Algorithm Competition," [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.

[8] Xilinx, Febuary 2009. [Online]. Available: www.xilinx.com/support/documentation/data_sheets/ds100.pdf.

[9] H. Sinan, "Hardware Evaluation of SHA-3 Candidates," Tech Polytechnic Institute and State University, Blacksburg, 2007.

[10] E. Biham, O. Dunkelman and Technion, "A Framework for Iterative Hash Functions — HAIFA," in *Second NIST Cryptographic Hash Workshop*, 2006.

[11] R. M. Secrecy, "Authentication, and public key systems," Stanford Ph.D. thesis, Stanford, 1979.

[12] J. P. Aumasson et al. , "SHA-3 proposal BLAKE (version 1.3)," 2009. [Online]. Available: http://www.131002.net/blake.

[13] Bernstein and J. Daniel, "ChaCha, a variant of Salsa20," in *SASC*, Lausanne, Switzerland, January 2008.

[14] Xilinx, March 16 2012. [Online]. Available:

www.xilinx.com/support/documentation/user_guides/ug190.pdf.

[15] Kris Gaj et al. , "Comprehensive Evaluation of High-Speed and Medium-Speed Implementations of Five SHA-3 Finalists Using Xilinx and Altera FPGAs," in *Third SHA-3 Candidate Conference*, Washinton, 2012.

[16] B. Baldwin et al. , "FPGA Implementations of the Round Two SHA-3 Candidates," in *FPL 2010*, 2010.

[17] S. Kerckhof et al. , "Compact FPGA Implementations of the Five SHA-3 Finalists," in *CARDIS 2011*, 2011.

[18] J. Bernhard, "Evaluation Of Compact FPGA Implementations For All SHA-3," in *Third SHA-3 Candidate Conference*, Washington, March, 2012.

[19] S. Nicolas and K. Paris, "BLAKE HASH Function Family on FPGA: From the Fastest to the Smallest," in *IEEE ISVLSI*, 2010.

[20] J. P. Kaps et al. , "Lightweight Implementations of SHA-3 Finalists on FPGAs," in *Third SHA-3 Candidate Conference*, Washington, March, 2012.

[21] L. Kashif, M. Rao. , A. Aziz. and M. Athar, "Efficient Hardware Implementations and Hardware Performance Evaluation of SHA-3 Finalists," in *Third SHA-3 Candidate Conference*, Washington, March 2012.

[22] J. L. Beuchat et al. , "Compact Implementations of BLAKE-32 and BLAKE-64 on FPGA," in *FPT 2010*, 2010.

[23] M. Knezevic, "Fair and Consistent Hardware Evaluation of Fourteen Round Two SHA-3 Candidates," in *Very Large Scale Integration (VLSI) Systems, IEEE Transactions*, May, 2012.

[24] D. Vaibhav, A. Richa and K. Y. Rajesh, "FPGA Based Area And Throughput Implementation of JH And BLAKE Hash Function," *IJCTT,* vol. 3, no. 2, p. 268, April 2012.

[25] NIST, "Federal Information Processing Standards Publication 180-1," April 1995. [Online]. Available: http://www.itl.nist.gov/fipspubs/fip180-1.htm.

[26] Xilinx, August 2012. [Online]. Available: www.xilinx.com/.../application_notes/xapp522-mux-design-techniques.pdf.

[27] E. Homsirikamol, M. Rogawski and K. Gaj, "Throughput vs. Area Trade-off in High-Speed Architectures of Five Round 3 SHA-3 Candidates Implemented Using Xilinx and Altera FPGAs," in *CHES 2011*, 2011.

[28] Knebl, D. Hans and Helmut, Introduction to Cryptography: Principles and Applications, Springer, 2007.

[29] S. Matsuo et al. , "How Can We Conduct "Fair and Consistent" Hardware Evaluation for SHA-3 Candidate," in *2nd SHA-3 Conference*, 2010.