Hardware Implementation of Secure Hash Algorithm(SHA-3) Candidate (Skein) on FPGA

Submitted by:

Muhammad Tariq

Supervisor:

Dr. Arshad Aziz

Thesis

Submitted to
Department of Electronic and Power Engineering,
College of Marine Engineering (PNEC), Karachi
National University of Sciences and Technology, H-12
Islamabad

In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
with Specialization in Communications

February 2011

IN THE NAME OF ALLAH,
THE MOST BENEFICENT, THE MOST MERCIFUL

Abstract

Cryptographic Algorithms are essential methods to attain security in computer and communication systems. In modern computer and communication sciences, fast and Efficient implementation (in terms of throughput and resouce usage) of these cryptographic algorithms is an active area of research. Hardware solutions for these cryptographic algorithms are often required to optimize the performance and to concentrate on security issues. Reconfigurable Platforms i.e. FPGAs (Field Programmable Gate Arrays) are ultimate choice among various hardware solutions because they combine flexibility, speed and resource efficiency. Skein is one of the 14 candidates which advanced in the round 2 of NIST SHA-3 competition. We choose it on the basis of its simplicity, security and speed. After the second round conference, Skein is among the 5 short listed candidates in round 3 of the competition. An efficient hardware implementation of Skein-256 algorithm have been presented in this thesis work and we have compared our results with other published hardware implementations of Skein. This paper investigates the performance characteristics of a high-speed hardware implementation of Skein on modern FPGA.

Acknowledgements

Above all, I am very much grateful to Allah Subhanaho Wa Ta'ala for His sympathies and blessings that He has always shower upon all His creatures.

I would like to thank all of my family members and other beloved ones for their support throughout my studies and thesis work by prayers and patience.

Then, I owe my deepest gratitude to my supervisor Dr. Arshad Aziz for his timely advice and guidelines. I mention here that, it was his motivation and encouragement that got me through many difficult situations during this thesis work.

It is a pleasure to thank all of Guidance Committee members, especially Dr. Athar Mahboob for their kind guidance with great patience and knowledge.

At last I am indebted to my many of my colleagues to support me during my studies and thesis work.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Motivation

Skein is one of the 14 candidates advanced to Round 2. Skein is a fast, versatile, and secure hash function. It is a family of Hash functions based on the tweakable block cipher Threefish and can be decomposed into the Threefish Block Cipher and the Unique Block Iteration (UBI) construction. In this thesis work, we have presented an efficient hardware implementation of Skein algorithm and compare our results with available implementations.

Cryptographic hash functions take an arbitrary block of data as input and output a fixed-size bit string, the (cryptographic) hash value. It is in this way that an accidental or intentional change to the message will change the hash value. The input is known as the **message**, and the hash value is known as the **message digest**. Cryptographic hash functions have many information security applications, particularly in digital signatures, message authentication codes (MACs). Moreover, these can be used for fingerprinting, data indexing in hash tables, to detecting duplicate data or to uniquely identify files, and as checksums to detect accidental data corruption during communication.

To ensure the long-term robustness of applications that use hash functions, National Institute of Standards and Technology (NIST) USA has announced a public competition on November 2, 2007 to develop a new cryptographic Hash algorithm called SHA-3. This competition is in response to recent advances in the cryptanalysis of commonly used hash algorithms. These algorithms include SHA family: SHA-0, SHA-1, SHA-256 and SHA-512, MD4 and MD5. In previous few years, cryptanalysis of these algorithms found serious vulnerabilities.

In response to NIST's announcement 64 submissions were reported by

the submission deadline of October 31, 2008. Among these, fifty-one entries were those which fulfilled the minimum submission requirements and were selected as the first round candidates in December 2008.

After presentation review and analysis in the first SHA-3 candidate conference in February 2009, these candidates reduced to 14 in Round 2 of the competition. A year was allocated for the public review, implementation and analysis of these algorithms.

The Second SHA-3 Candidate Conference was held on August 23-24, 2010 in University of California, Santa Barbara. Five short listed candidates, advanced in round 3 are BLAKE, Grostl, JH, Keccak and Skein. The tentative timeline for the end of this competition and selection of the successful candidate for SHA-3 is in 4th quarter of 2012 [2].

For our research work, we have choose fast and efficient implementation of Skein algorithm. Hardware solutions for these algorithms are often required to optimize the performance as well as to address physical security issues. Re-configurable Platforms i.e. FPGAs (Field Programmable Gate Arrays) are ultimate choice among various hardware solutions to combine flexibility, speed and resource efficiency.

1.2 Thesis Scope

In this thesis work, efforts are made to do an efficient hardware implementation of SHA-3 candidate **Skein** proposed by Bruce Schneier on FPGA, in terms of performance. This work may be used as a contribution in finalizing the candidate of new Secure hash Algorithm SHA-3. Out of two major hardware description languages used in industry, VHDL and Verilog HDL, we choose Verilog as Hardware Description Language (HDL) to implement the design. FPGA devices from two major vendors, Xilinx and Altera, dominate the market with about 90 percent of the market share. Out of these two vendors, Xilinx leads, so we feel that, it is appropriate to focus on FPGA devices from Xilinx. We used two Xilinx FPGA architectures Virtex 5 and Spartan 3 for implementation and comparison of results.

In order to compare and validate the hardware results the algorithm is also implemented in software using C language prior to hardware implementation. Software implementation also helped us in understanding the algorithm and gaining confidence.

1.3 Chapter Organization

Chapters of the thesis are organized as follows: Chapter 2 gives some background knowledge of cryptographic hash functions and discusses essential properties and applications of cryptographic hash functions. Moreover, it also defines the requirement of the new secure hash algorithm (SHA-3). Chapter 3 covers description about secure hash algorithm SHA-3 candidate **Skein** Algorithm in detail. Chapter 4 describes the background knowledge of FPGA and its advantages over the ASIC designs. This chapter also summarizes internal architecture of Xilinx's FPGAs. Chapter 5 describes the methodology adopted for this work and general design considerations for software and hardware implementation. Chapter 6 explains the software implementation of the **Skein** algorithm. Chapter 7 exclusively explains the hardware design implementation of the Skein algorithm. In Chapter 8 we provide the results of our work and compare it with some well known implementations. Finally, Chapter 9 concludes the work experience and defines future directions to extend and improve the existing achievement.

Chapter 2

CRYPTOGRAPHIC HASH FUNCTIONS

2.1 Introduction

A cryptographic hash function is a conclusive method whose input is random block of data and output is a fixed-size bit string, which is known as the (Cryptographic) hash value. It is in this way that a fortuitous or intentional change to the message will change the hash value. In general terms the ideal cryptographic hash function has following major or significant properties:

- It should be very difficult to find two inputs that produce the same message digest.
- It is easy to compute the hash value for any given message.
- It is very hard to find a message that has a given hash.
- It is infeasible to modify a message without changing its hash.

2.2 Properties

Now we more formally describe the various properties of a cryptographic hash functions.

2.2.1 Preimage Resistance

Preimage resistance is the measure of difficulty to find an input message from which a given hash value is computed. This means that hash functions

are one-way functions. Functions that lack this property are vulnerable to preimage attacks.

If we have given a hash value ***h*** it should be hard to find any message ***m*** such that

$$h = hash(m)$$

2.2.2 Second Preimage Resistance

Given an input message ***m1***, second preimage resistance is the measure of difficulty to find another input message ***m2*** such that both messages have the same hash value. i.e.

$$hash(m1) = hash(m2)$$

where m1 $\neq$ m2

This property is sometimes referred to as weak collision resistance, and functions that lack this property are vulnerable to second preimage attacks.

2.2.3 Collision Resistance

Collision resistance is the measure of difficulty to find two different messages ***m1*** and ***m2*** such that:

$$hash(m1) = hash(m2)$$

Such pair of messages is called a cryptographic hash collision, a property which is sometimes referred to as strong collision resistance.

These properties means that without changing hash value one cannot modify input message. So it implies that if two messages have same hash value, they are identical.

2.3 Hash Functions are not Encryption

Hash functions and encryption are different and it is important to understand their difference. Encryption converts plaintext into ciphertext using a key and by using the appropriate key it converts it back. The two texts roughly correspond to each other with respect to size. This means that small plaintext yields small ciphertext and so on. "Encryption" is a two-way or reversible operation.

On the other hand, hash function converts a stream of data into a fixed size hash value. No matter how long the message is but its hash value will be of fixed size and it is strictly a one way operation.

2.4 Applications

Cryptographic hash functions have many information security applications, particularly in digital signatures, message authentication codes (MACs), and other types of authentication [3]. Moreover these can be used for fingerprinting, for data indexing in hash tables, to detect duplicate data or uniquely identify files, and as checksums to detect accidental data corruption during communication.

2.4.1 Verifying File Integrity

Verifying file integrity is an important use of hash functions. When large files are made available for download during transmission on networks, sometimes an error may occur corrupting the file. To verify file integrity, websites publish the original hash values of their download bundles. The recipient finds out the hash of the received file and compares it with the original hash value, to confirm that the received file is identical to the original.

2.4.2 Hashing Passwords

It is not a good idea to store password in their original form. It is because somebody may get access to the database where they are stored. Therefore, rather than storing the password itself, it is more protected to store hash value of that password. Hashes are one way functions and are not reversible so it is not feasible to find the actual password even after getting access to the stored hash values.

2.4.3 Digital Signatures

A digital signature is a way to ensure the identity of the sender of a message or the signer of a document. Moreover, it also guarantees that the message which arrived at the receiver is intact. To ensure authentication digital signatures are based on certain types of encryption. Digital signatures cannot be duplicated and they are easily transportable.

Figure 2.1 illustrates that how the signature is represented and how does one know whether a digital signature applies to a particular document or not: It is obvious from the figure that instead of signing the message digitally; the hash value of the message is signed. It is because computing digital signature of long message is time consuming. To verify the signature the recipient calculates the hash of the message and compares it with the fingerprint that was signed. If they are the same, the received message is authentic.

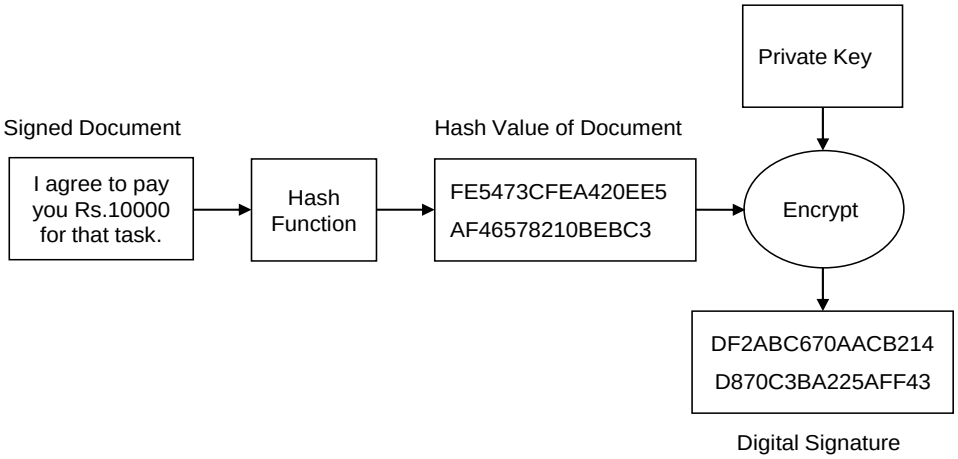


Figure 2.1: Digitally signing a document

2.5 Requirement of New Secure Hash Algorithm SHA-3

There is a long list of cryptographic hash functions, although many have been found to be vulnerable and should not be used. A successful attack against a weakened variant of an algorithm destabilizes the experts' confidence, although if a hash function has never been broken, which lead to its rejection. Such type of vulnerabilities were found in a number of hash functions in August 2004 that were popular at the time, including SHA-0, RIPEMD, and MD5. This leads to the long-term security of SHA-1, RIPEMD-128, and RIPEMD-160 algorithms susceptible. SHA-1 is strengthened version of SHA-0, while RIPEMD-128, and RIPEMD-160 are both strengthened versions of RIPEMD.

As of 2009, the two most commonly used cryptographic hash functions are MD5 and SHA-1. MD5 (Message Digest Algorithm 5) is the hash function designed by Ron Rivest [4] in 1991 as a strengthened version of MD4 and SHA-1 hash function was developed by the NSA.

In 2004, Xiaoyun Wang et al [5] [6] presented the collision for the full MD5. Moreover there was a breakthrough in cryptanalysis of SHA-1 hash Algorithm in August 2005. Professor Xiaoyun Wang and co-authors found that it is possible to find a collision in SHA-1 in 2^{63} [7]. Previously it was thought that 2^{80} operations are required to find a collision in SHA-1 for a 160-bit hash function. This attack is expected to find two different messages having the same hash value i.e. the hash collision in 2^{63} operations.

Suppose we have a hash function that produces an n-bit long hash value or message digest and we are trying to find some message which will produce a particular message digest value y, then because each message digest value is equally likely, we expect to have to try 2^n possible input values. If we are trying to find a collision, then by the birthday paradox, we would expect that after trying $2^{n/2}$ possible input values we would have some collision. Van Oorschot and Wiener [8] showed how such a brute-force attack might be implemented.

Although no attacks have yet been reported on the SHA-2 variants, they are algorithmically similar to SHA-1. However, to ensure the long-term robustness of applications that use hash functions National Institute of Standards and Technology (NIST) has announced a public competition in the Federal Register Notice published on November 2, 2007 [9] to develop a new cryptographic hash algorithm called SHA-3. This competition is to be finalized by the year 2012 [2].

Chapter 3

SHA-3 CANDIDATE (SKEIN) HASH ALGORITHM

3.1 Introduction

Skein is one of the 14 candidates advanced to round 2. Skein is a fast, versatile, and secure hash function. Skein is a family of hash functions based on the tweakable block cipher Threefish and can be decomposed into the Threefish Block Cipher and the Unique Block Iteration (UBI) construction. The block and key size of the Threefish are equal and may be of 256, 512, or 1024 bits.

3.2 Description

Skein uses three simple operations of 64-bit adders along with shift, and XOR to create the output message digest. It can be divided into two parts from the implementation point of view.

- Threefish Block Cipher
- Unique Block Iteration (UBI) construction

3.2.1 Threefish Block Cipher

Skein's compression function is based on Threefish which is a large tweakable block cipher [4]. The block and key size of Threefish are equal and can be set to 256, 512 or 1024 bits, and they are designated as: Threefish-256, Threefish-512, and Threefish-1024 respectively. Threefish structural design consists of round operations. Threefish-256 and Threefish-512 compression

function is made of 72 consecutive round operations while the Threefish-1024 requires 80 rounds. Each round of the Threefish block cipher is made of two instances of a Mix function along with a permutation module while a round key is added to the data before the first round and after each 4 consecutive rounds as shown in Figure 3.1

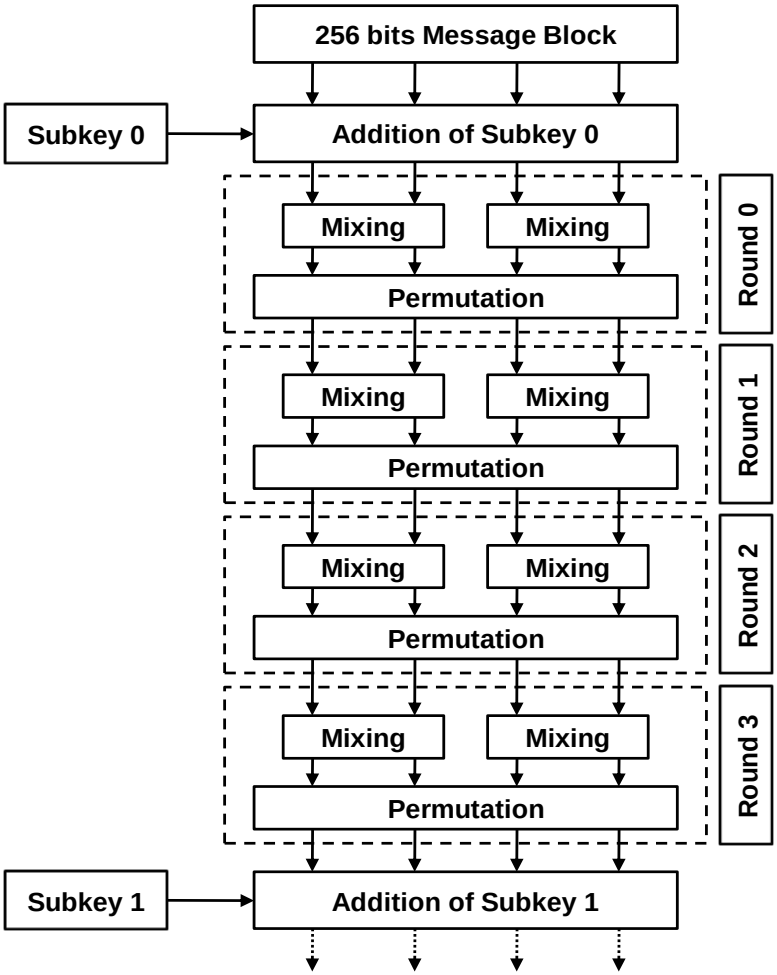


Figure 3.1: First four round operations of the Threefish-256 cipher

The mix operation consists of addition modulo 2^{64}, XORs and left-rotates. These operations are defined on the intermediate state organized in 64-bit words. The MIX operation transforms two of these 64-bit words and is common to all Threefish variants as shown in Figure 3.2:

Mix function has two input words (X_0 and X_1) and produces two output

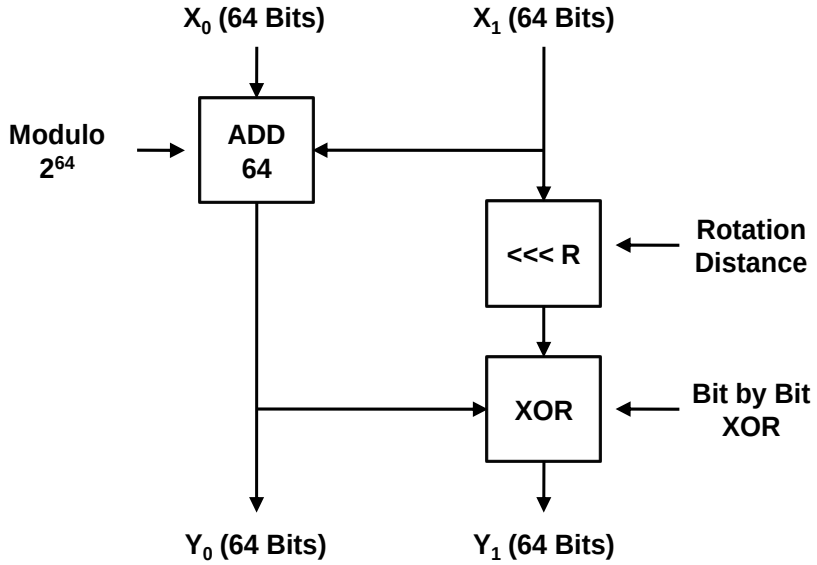


Figure 3.2: Threefish MIX operation

words (Y_0 and Y_1) using the following relations:

$$Y_0 = (X_0 + X_1) mod 2^{64}$$

$$Y_1 = (X_1 <<< R) \oplus Y_0$$

Where $\oplus$ is the bit by bit XOR operation and $<<<$ is the left rotate operator and R (Rotation Distance) is a constant value depends on the Threefish block size, the round index and the position of the two 64-bit words in the Threefish block [10]. All Threefish rounds are similar apart from rotation constant in mixing operation. Rotation constants for different Threefish variants are shown in Figure 3.3[10].

The subsequent permutation operation reorders 64-bit words constructed from a Threefish block. Values for word permutation are fixed for each Threefish variant as shown in Figure 3.4.

Subkeys or round keys are consisting of three contributions: an input key words, tweak words, and a counter value. The key schedule turns the key and tweak into a sequence of subkeys, each of which equal to the size of the block.

Tweak depends upon number of factors including position and the bit length of message block. Tweak is contructed as shown in Figure 3.5

Nw	4		8				16							
j	0	1	0	1	2	3	0	1	2	3	4	5	6	7
0	14	16	46	36	19	37	24	13	8	47	8	17	22	37
1	52	57	33	27	14	42	38	19	10	55	49	18	23	52
2	23	40	17	49	36	39	33	4	51	13	34	41	59	17
3	5	37	44	9	54	56	5	20	48	41	47	28	16	25
d = 4	25	33	39	30	34	24	41	9	37	31	12	47	44	30
5	46	12	13	50	10	17	16	34	56	51	4	53	42	41
6	58	22	25	29	39	43	31	44	47	46	19	42	44	25
7	32	32	8	35	56	22	9	48	35	52	23	31	37	20

where:
Nw is number of 64-bit words in message block.
j is the number of 64-bit input word in mixing operation.
d is the round number.

Figure 3.3: Rotation Constant (R)

	i th word															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	0	3	2	1												
Nw= 8	2	1	4	7	6	5	0	3								
16	0	9	2	13	6	11	4	15	10	7	12	3	14	5	8	1

Figure 3.4: Values for the word permutation

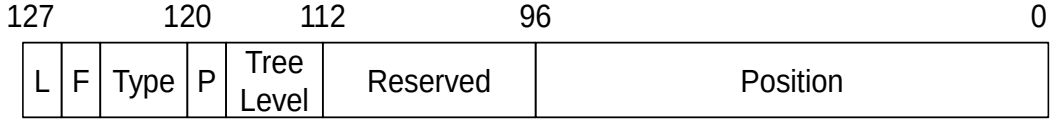


Figure 3.5: The fields in the tweak value

Table 3.1: The fields in the tweak value

Name	Bits	Description
Position	0-95	Number of bytes of message processed so far (including this block)
Reserved	96-111	For future use; must be 0
TreeLevel	112-118	Level of the tree when tree hashing is used; 0 for non-tree computations
BitPad	119	1 if the message block is padded and 0 otherwise
Type	120-125	The application-specific UBI function being performed
F (First)	126	1 for first block of message and 0 otherwise
L (Last)	127	1 for last block of message and 0 otherwise

Skein has many possible parameters. Each parameter, whether optional or mandatory, has its own unique type identifier and value. Type values are in the range $0 \ldots 63$. Skein processes the parameters in numerically increasing order of type value, as listed in Table 3.2.[10]

Table 3.2: Values for the type field

Symbol	Value	Description
Key	0	Information hashed is a key (for MACs and KDFs)
Cfg	4	For computing the configuration block (initialization vector)
Prs	8	For personalized hashing
PK	12	Personalization using a public key (for digital signature hashing)
Kdf	16	Key identifier
Non	20	Nonce (for stream cipher or randomized hashing)
Msg	48	Message that Skein is hashing
Out	63	For computing Skein output

3.2.2 Unique Block Iteration (UBI) construction

The UBI construction is a variant of the Cascade or (Merkle-Damgard) construction. It uses a tweakable block cipher in Matyas-Meyer-Oseas mode to

form a compression function, and uses the bit offset of the block being hashed as the tweak. Figure 3.6 shows an example of UBI mode.

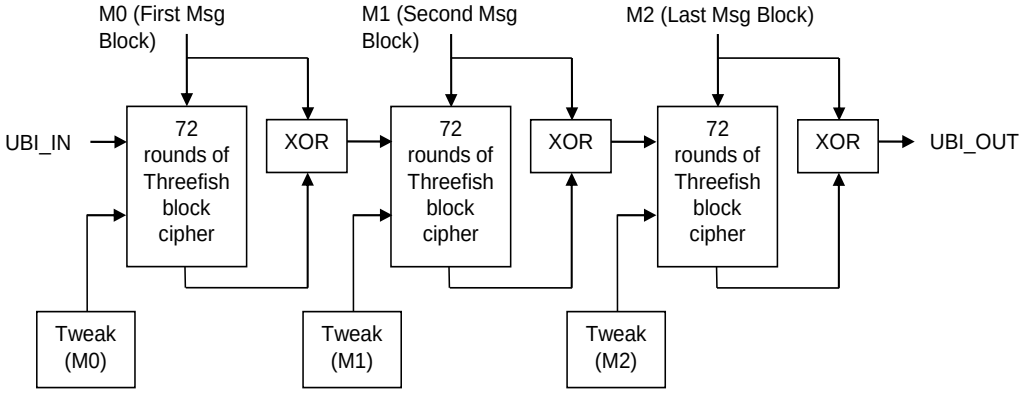


Figure 3.6: Unique Block Iteration (UBI) mode

It is supposed that the message M comprises of three message blocks(M0, M1 and M2). UBI_IN is the first Threefish encryption key which is used along with the tweak value for the encryption of first message block. The output of the Threefish block cipher is XORed with message block itself and its output along with new tweak value is used for the encryption of the next block of message. It means that a new key is used for the encryption of each block. As mentioned above that the tweak values depend on the position and bit length of the respective message block. UBI is used in Skein not only for compression and the output transformation, but also for other optional operation modes (e.g. tree hashing, keyed hashing).

Chapter 4

FIELD PROGRAMMABLE GATE ARRAY

4.1 Introduction

In the family of reconfigurable hardware Field Programmable Gate Array (FPGA) is one of the most pronounced name. FPGAs are reprogrammable silicon chips. One can configure these chips using pre-built logic blocks and programmable routing resources to implement custom hardware functionality in relatively short time and very low cost as compared to ASICs. We can develop digital computing tasks in software and compile them down to a configuration file or bit stream that contains information on how the components should be wired together. In addition, FPGAs are completely reconfigurable and right away take on a brand new personality when we recompile a different configuration of circuitry. FPGA chip adoption across all industries is driven by the fact that FPGAs combine the best parts of processor-based systems and ASICs. FPGAs provide hardware-timed reliability and speed, but they do not require mass production to justify the large upfront expense of custom ASIC design. Reprogrammable silicon also has the same flexibility of software running on a processor-based system, but it is not limited by the number of processing cores available. Unlike processors, FPGAs are truly parallel in nature so different processing operations do not have to compete for the same resources. Each independent processing task is assigned to a dedicated section of the chip, and can function autonomously without any influence from other logic blocks. As a result, the performance of one part of the application is not affected when additional processing is added.

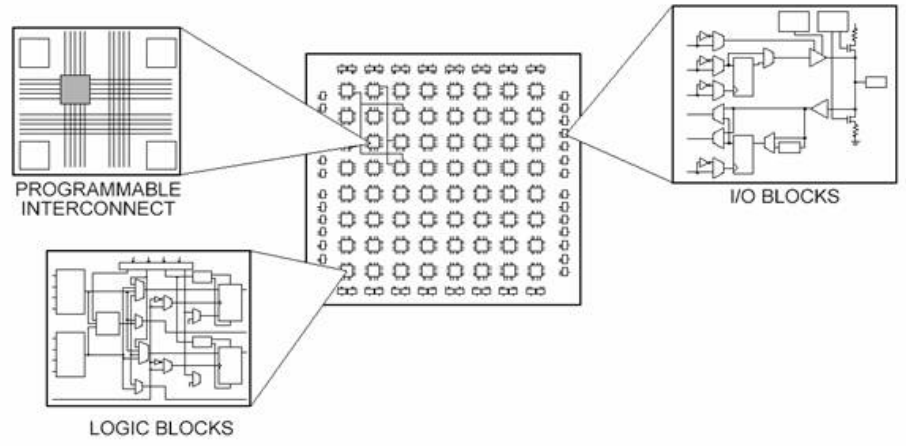


Figure 4.1: Different Parts of FPGA

4.2 FPGA's Scope in Industry

FPGA technology continues to gain momentum, and the worldwide FPGA market is expected to grow from 1.9 billion in 2005 to 2.75 billion by 2010 [11]. Since its invention by Xilinx in 1984, FPGAs have gone from being simple glue logic chips to actually replacing custom Application-Specific Integrated Circuits (ASICs) and processors for signal processing and control applications.

4.3 Advantages of FPGA

National Instruments published an excellent article about the advantages of FPGAs [12]. These advantages are being summarized as follows:

Performance: FPGA provides faster response time due to hardware level control of I/Os. Moreover, taking advantage of hardware parallelism, FPGAs exceed the computing power of digital signal processors (DSPs) by breaking the paradigm of sequential execution and accomplishing more per clock cycle. BDTI, a noted analyst and benchmarking firm, released benchmarks showing how FPGAs can deliver many times of processing power per dollar of a DSP solution in some applications [13].

Time to Market: FPGA technology offers flexibility and rapid prototyping capabilities in the face of increased time-to-market concerns. You can

test an idea or concept and verify it in hardware without going through the long fabrication process of custom ASIC design [14].

Cost: As system requirements often change over time, the cost of making incremental changes to FPGA designs are quite negligible when compared to the large expense of respinning an ASIC.

Reliability: For any given processor core, only one instruction can execute at a time, and processor-based systems are continually at risk of time-critical tasks pre-empting one another. FPGAs, which do not use operating systems, minimize reliability concerns with true parallel execution and deterministic hardware dedicated to every task.

Long-term Maintenance: FPGA chips are field-upgradable and do not require the time and expense involved with ASIC redesign. Being reconfigurable, FPGA chips are able to keep up with future modifications that might be necessary. As a product or system matures, we can make functional enhancements without spending time redesigning hardware or modifying the board layout.

4.4 Xilinx FPGA

The Spartan-3 family architecture consists of five fundamental programmable functional elements [1]:

Configurable Logic Blocks (CLBs) : CLB's contain flexible Look-Up Table (LUTs) that implement logic plus storage elements used as flipflops or latches. CLBs perform a wide variety of logical functions as well as store data.

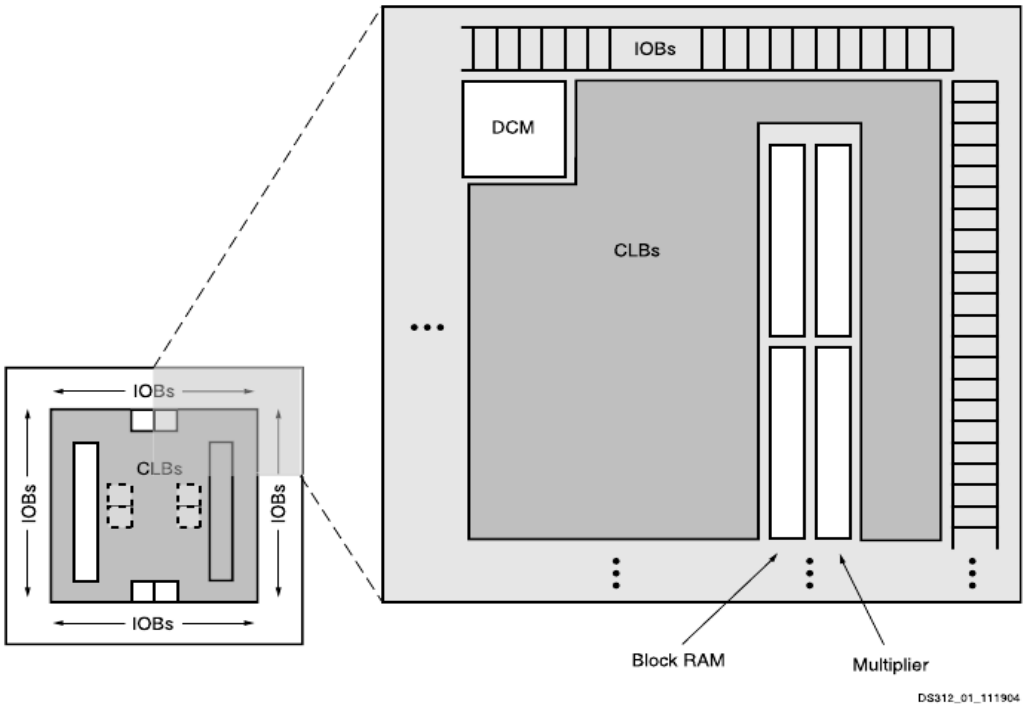
Input/Output Blocks (IOBs) : IOBs control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. IOBs support a variety of signal standards including four high-performance differential standards.

Block RAM (BRAM) : Block RAM provides data storage in the form of 18-Kbit dual-port blocks.

Multiplier blocks : Multiplier blocks accept two 18-bit binary numbers as inputs and calculate the product.

Digital Clock Manager (DCM) blocks : DCM blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase shifting clock signals.

These elements are organized as shown in Figure 4.2. A ring of IOBs surrounds a regular array of CLBs. Each device has two columns of block RAM. Each RAM column consists of several 18-Kbit RAM blocks. Each block RAM is associated with a dedicated multiplier. The DCMs are positioned in the center with two at the top and two at the bottom of the device. The Spartan-3E family features a rich network of traces that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing.



Notes:

1. The XC3S1200E and XC3S1600E have two additional DCMs on both the left and right sides as indicated by the dashed lines. The XC3S100E has only one DCM at the top and one at the bottom.

Figure 4.2: Spartan-3E Family Architecture [1]

Chapter 5

GENERAL DESIGN CONSIDERATIONS

5.1 Introduction

This chapter includes a discussion about general design consideration and design flow of the thesis. The discussion starts from the software implementation of the algorithm in high level language to the hardware implementation. Moreover there is a discussion about the tools which are used for designing purpose.

5.2 Design Flow

The thesis was completed using following design flow:

1. **Background Knowledge:** This step includes study of the various algorithms proposed by different candidates for SHA-3 competition. We choose Skein proposed by Bruce Schneier for our work because of its simplicity, speed and security.

2. **Software Implementation:** The algorithm which was chosen in previous step i.e. step 1 is implemented in a high level language i.e C language.

3. **Architecture Designing:** Designing of a suitable architecture appropriate for hardware implementation.

4. **Hardware Implementation:** By using hardware description language (HDL) i.e Verilog, the architecture which is designed is implemented.

5. **Simulation and Verification:** Verification of the hardware design through behavioral simulation and comparing results with software implementation at step 2.

6. **Synthesize the Hardware:** A netlist of the design depends on different technologies is used as a input to field programable gate array implementation. The generation of netlist is the synthesization of the hardware.

7. **Simulation and Verification:** By comparing the software results, again simulate the design and verify it at gate level.

8. **Optimization:** Design optimization.

5.3 Background Knowledge

Among a number of submissions to NIST only 14 advanced to round 2. The first step involved is to choose an algorithm for implementation. We choose Skein for our work because of its simplicity, security and speed . In order to know about the general design considerations and chose a best architecture, it is very significant to thoroughly study about the available implementations. There exists very few such implementations particularly on FPGA. Two of which studied in detail are [15] and [16].

5.4 Software Implementation

In order to get confidence and fully understand the algorithm of chosen candidate, software implementation using a high level language is very beneficial. Additionally, it is also advantageous in verifying results achieve by hardware implementation. Initially skein-256 was implemented using C language then it is upgraded to Skein-512. Software implementation includes simple and basic operations i.e. addition, XOR and left shift.

5.5 Architecture Designing

From the designing point of view it is a good practice to separate a design architecture into control and data paths. Data path contains set of registers and combinational logics it is responsible for register load/unload and computational operations. Control path consists of finite state machine (FSM), counter, state registers. FSM is implemented through state encoding and it

is used for controlling various operations of data path. The control path and data path are interfaced to each other through status and control signals.

5.6 Hardware Implementation

There may be two design methodologies to implement a design on to hardware: combinational and sequential. In our work we implemented hardware using both sequential and combinational methodologies and compare our results with available implementations for each of them. The time required to process input's first bit to the output's last bit is the measure of the maximum operating frequency in combinational designs. The maximum operating frequency is defined by the critical path length of a design. On the other hand, the maximum combinational path delay between two registers defines the critical path length in sequential design.

5.7 FPGA Design Flow

A typical FPGA Design Flow is shown in 5.1:

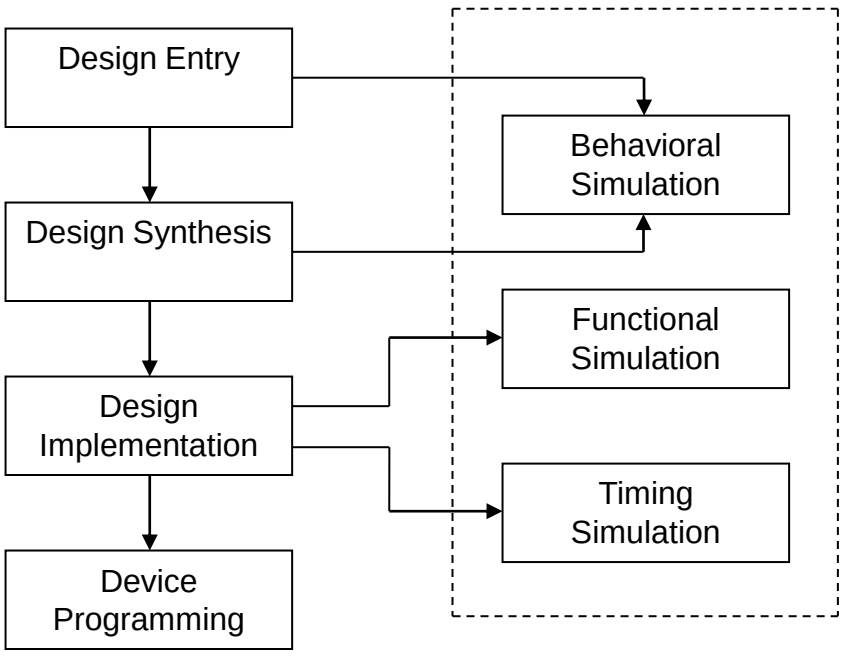


Figure 5.1: FPGA Design Flow

Design Entry

Design entry is the first step in FPGA design flow. To enter a design source files are created. Different formats are used to create source files which includes a Hardware Description Language (HDL) e.g. VHDL and Verilog or a schematic. A FPGA design includes a top-level source file and a number of lower-level source files in any format of the format i.e. either a schematic or a HDL file. For design entry Verilog is used in this research work.

Behavioral Simulation

The design is tested in terms of its logical functionality or behavior of the design by the help of a test bench which is defined by a user. Again different formats such as a Hardware Description Language (HDL) or a schematic may be used to define a test bench.

Design Synthesis

Technology dependant netlist files are created from a number of source files in design synthesis step. The netlist files acts as input to the implementation module and the gate level architecture is defined by these netlist files. The design may be optimized either for minimum area utilization or for better throughput. This option is provided while mapping the design at the gate level by various synthesis tools.

Design Verification (Simulation)

The functionality and behavior of a design verified by functional simulation, while timing is verified by timing simulation of the design. Implementing the circuit in FPGA defines the actual placement and routing of the circuit and timing simulation need to know it so we do the timing simulation after implementing the design to find the exact speed and timing of the circuit. Design verification is very important step that should be done at different stages of the design.

Design Implementation

After synthesis step in which netlist file is generated, the design implementation step converts the logic design into a physical file. This physical file can be downloaded on the target device (e.g. Spartan or Virtex FPGA). Design implementation includes these three sub-steps: *Translating* the netlist, *Mapping* and *Place and Route*.

Device Programming

Device programming is the downloading of the programming file to the target device. It is the actual configuration of the FPGA device.

Chapter 6

SOFTWARE IMPLEMENTATION

6.1 Introduction

We implement Skein-512-512 using C, that is the block size and hash value both are of 512 bits. Presently only few software implementations of Skein are available, using java, .net, C♯ etc. According to the best of our knowledge no software implementation using C language is available.

6.2 Key and Tweak Extension

In our implementation prior to the key schedule module, extended key and extended tweak are calculated by the help of key preprocessing circuit as shown in Figure 6.1.

The IVs, t0 and t1 are the seed for key scheduling, specified by message input. The preprocessing circuit yields k0 . . . k8 and t0 . . . t2.

6.3 Subkey Schedule Module

Sequence of subkeys is generated based on k0 . . . k7 and t0 . . . t1. Initial key k0 . . . k7 is provided for the first time only, the hash value calculated for each block, acts as key for the next block of message. Tweak value is calculated for each block which depends upon the offset of the message block and bit length of the message block. On the basis of these two i.e. key and tweak, key schedule module calculates round keys prior to processing each block as shown in Figure 6.2:

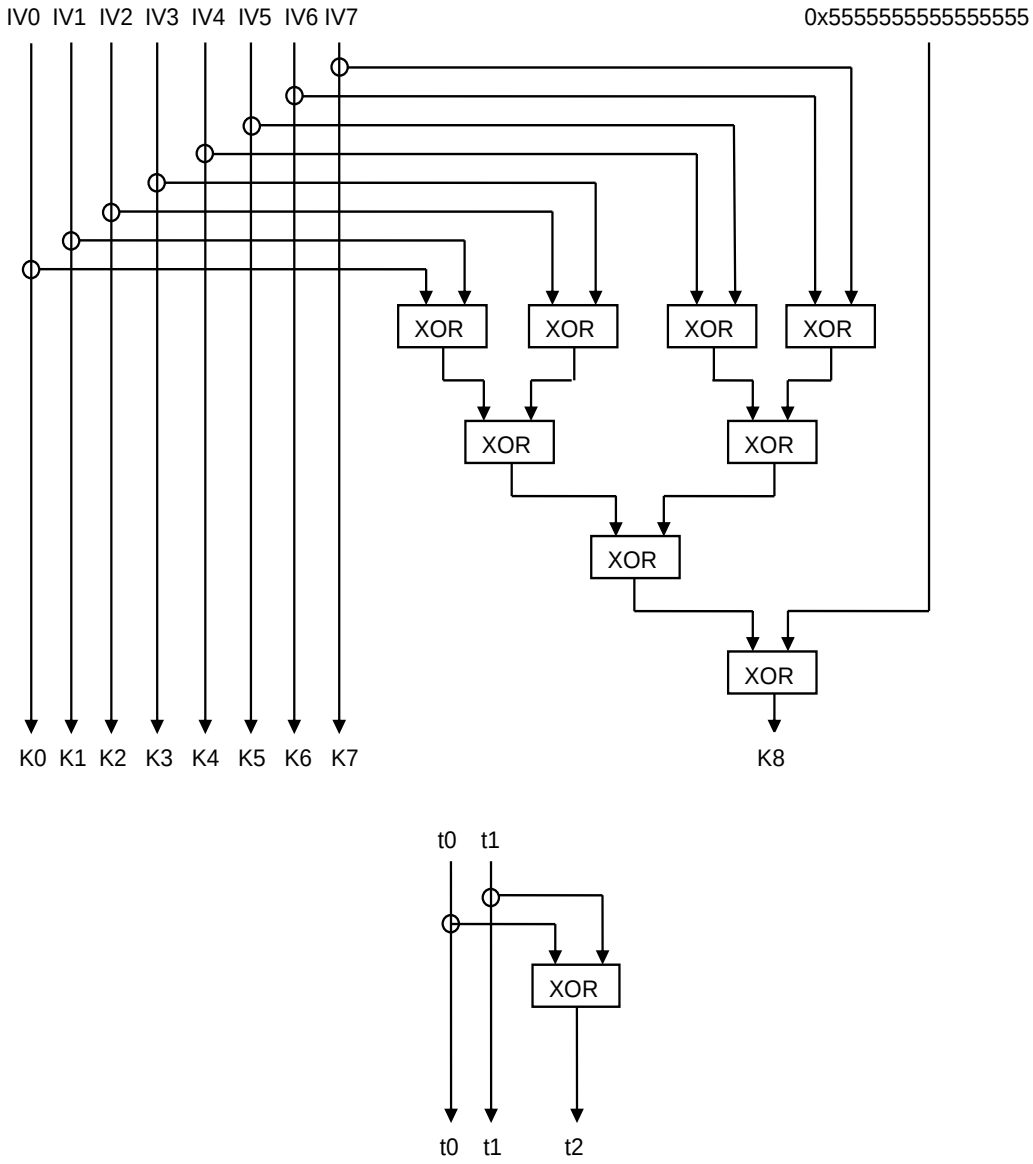


Figure 6.1: Preprocessing Circuit for Skein-512

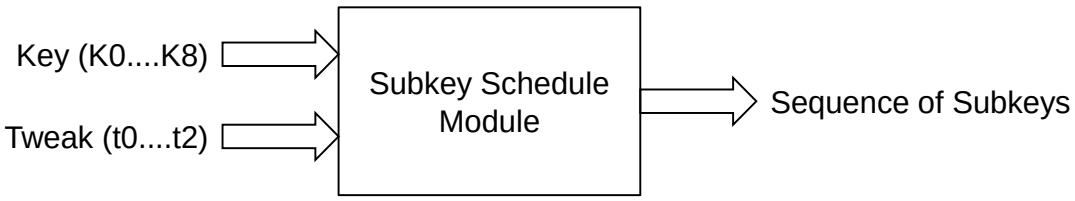


Figure 6.2: Key Scheduling

These round keys are stored as two dimensional array. Key schedule module basically comprises of addition of 64 bit words modulo 2^{64}. Each subkey is a combination of all but one of the extended key words, two of the three extended tweak words, and the subkey number.

6.4 Threefish Operation

After key schedule all 72 rounds of Threefish operation are executed. Each Threefish round comprises of Mix and Permutation operations. A Subkey is added after every four consecutive rounds. Mix operation includes three basic operations: Addition of 64 bit words modulo 2^{64}, 64 bit words XOR operation and left rotate operation as described in chapter 3. The value of the rotation distance R is a constant value depends on the Threefish block size, the round index and the position of the two 64-bit words in the Threefish block. According to these three parameters value of R is given in Skein hash Function Family version 1.2 submitted to NIST [10]. The rotation constants for the MIX operation repeat after eight rounds. Therefore for the implementation of Skein-512, an array of 8 x 4 is used. The subsequent permutation operation reorders 64-bit words constructed from a Threefish block. Values for word permutation are fixed for each Threefish variant. In software implementation these values are given in an array.

6.5 UBI Chaining

After all 72 rounds of Threefish operation the output of threefish is XORed with current block of message being Hashed. This is required for UBI chaining. The output of the Unique Block Iteration construction is either the required hash value or it is used as key value to process the next [17] block of message.

The overall processing of the software implementation of a block of message is shown in Figure 6.3

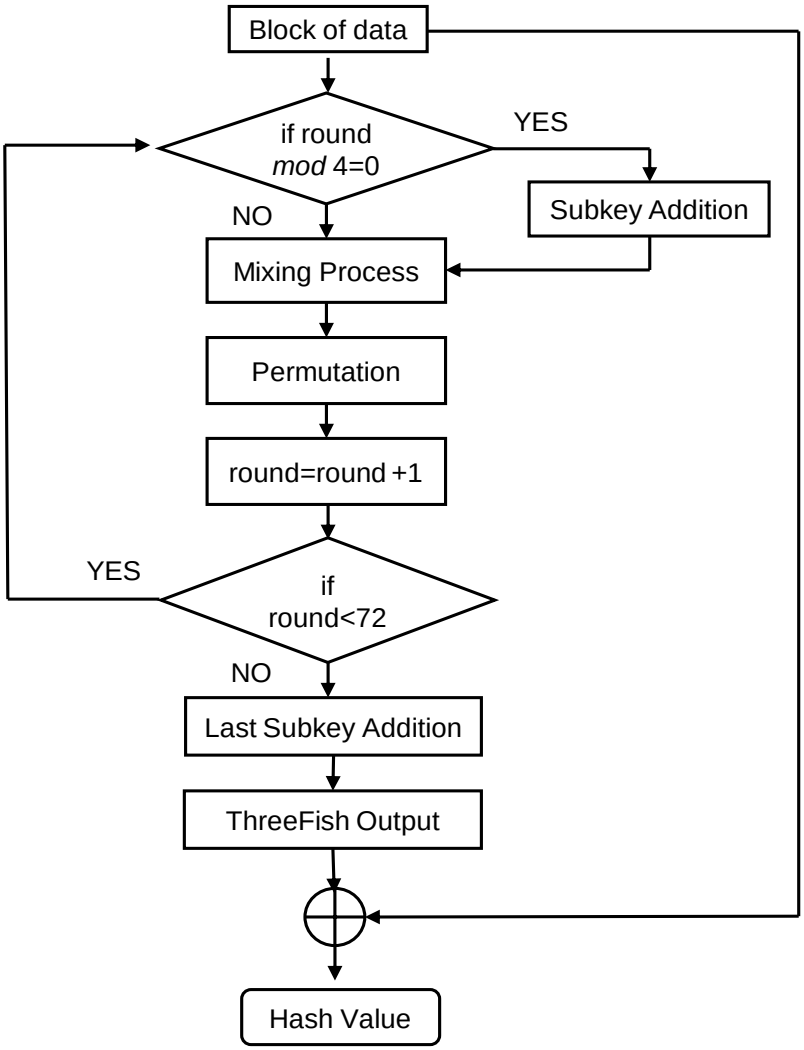


Figure 6.3: Software Implementation Flow chart

6.6 Result

This implementation calculates the hash value of a given message at a speed of 4.9 Mbytes per second on Intel Core2Duo 2GHz processor.

Chapter 7

HARDWARE IMPLEMENTATION

We have implemented the core functionality of Skein basic variant from scratch i.e. Skein-256-256 and used two different architectures for hardware module. One is combinational and the other is sequential design.

7.1 Combinational Design

Combinational hardware architecture design is shown in Figure 7.1. The block of input message is stored inside a 256 bit input register(In_Reg). Moreover tweak value and initial key are also stored inside two different registers of 128 bits and 256 bits respectively. **Subkey Schedule Module**generates sequence of subkeys (subkey0-subkey18) on the basis of tweak value and initial key. Each subkey is 256 bits long. First subkey(subkey0) is added to the original message block in a 256 bit adder then after every four consecutive rounds each of the subkeys is added to the result of the previous round. In this way for 72 round of Skein-256, 19 subkeys are required. The round process consists of Mix and Permute operations. 256 bits message is divided into four 64 bits words as defined in section 3.2.1 and Figure 3.1. After all 72 rounds there exists a one level of XOR gate, to create the UBI construction of the algorithm. Final hash value is stored in a 256 bit output register(Out_Reg).

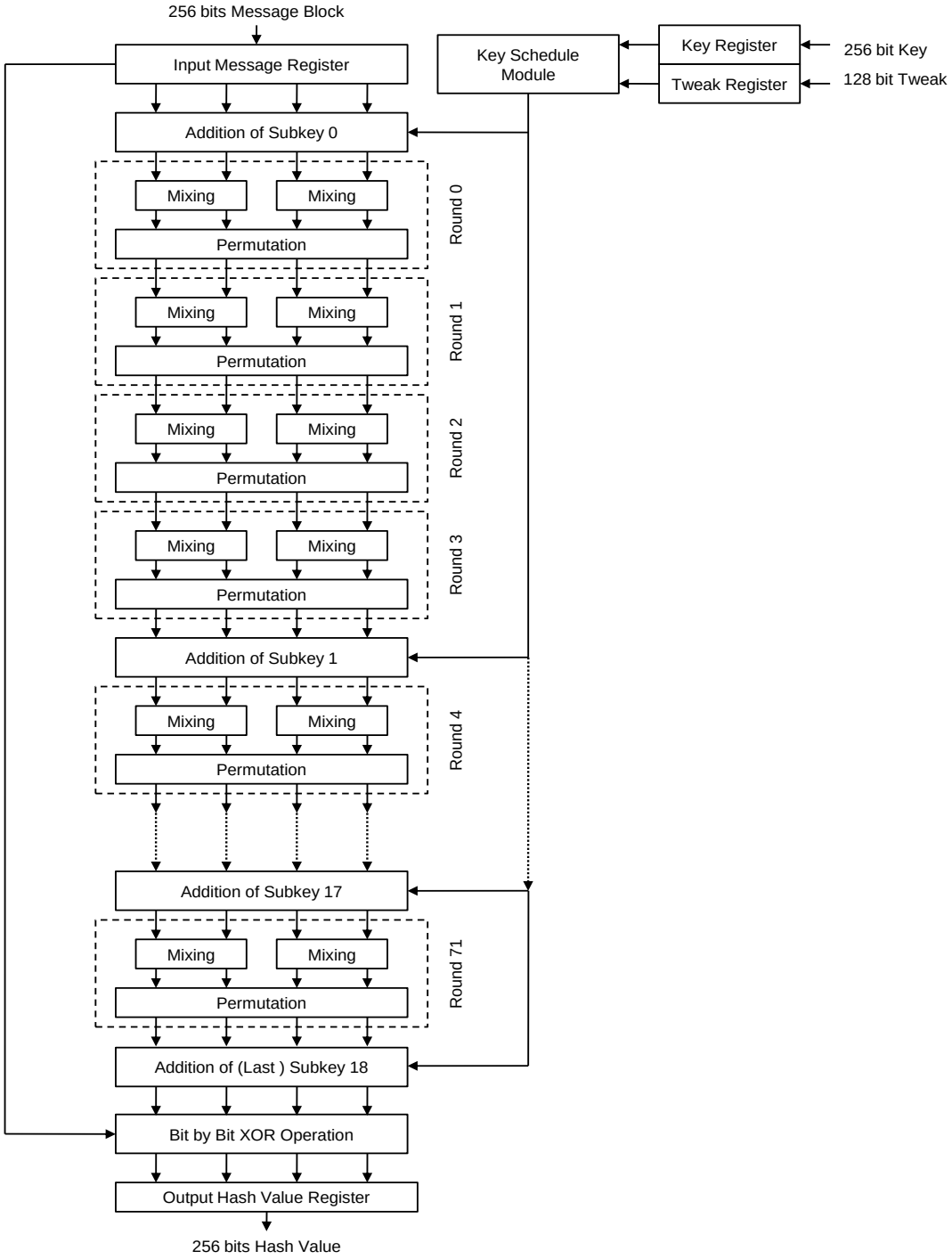


Figure 7.1: Hardware architecture used for compression function of Skein-256

7.2 Sequential Design

7.2.1 Data and Control Paths

The implementation can be divided into control and data paths. Control path consists of Finite State Machine, State register, clock and counter. Data path consists of Input and Output registers, key and tweak registers, Key Schedule module, Round_E and Round_O modules and Add_Subkey module as shown in Figure 7.2. Input message is stored in In_Reg and output hash value is stored in Out_Reg.

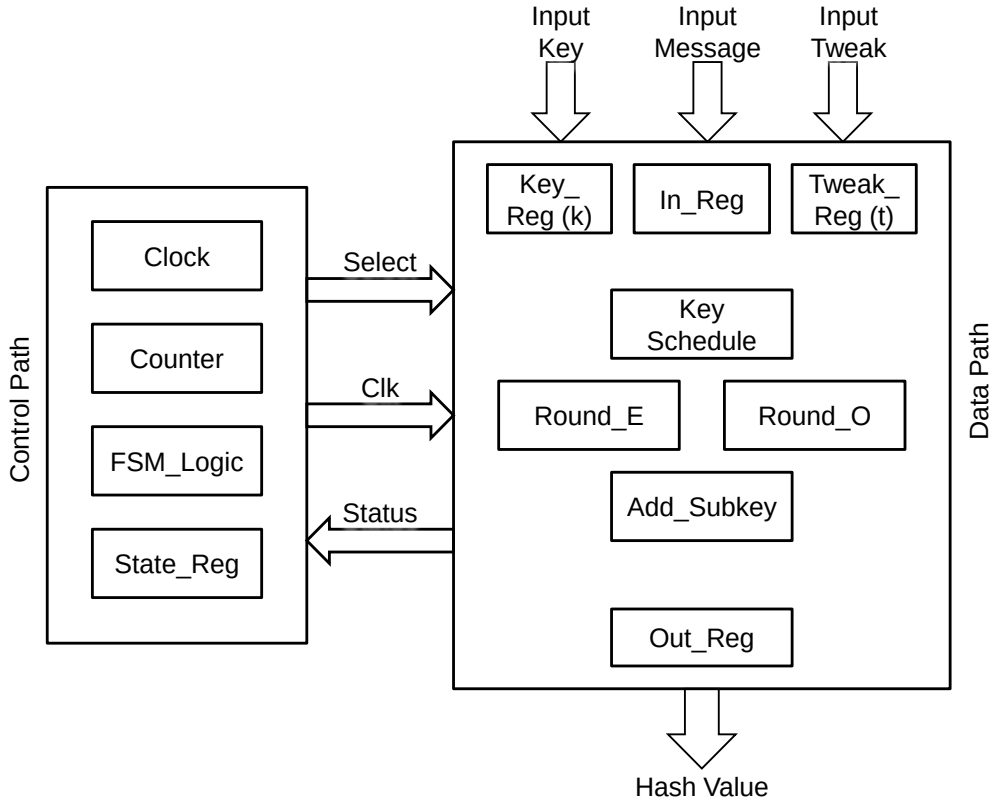


Figure 7.2: Hardware architecture seperated in Control and Data paths

7.2.2 Add_Subkey module

Add_Subkey module is a 256 bit adder having inputs from key schedule module and Mux_1. Select input (S1) of the Mux_1 is at logic 0 only for the first clock cycle and pass the original message block to Add_Subkey module to

add it with subkey0. After first clock S1 remains at high logic and pass the result of the previous round to add it with the next subkey. Output of Add_Subkey is used as input of Demux_1. Demux_1 and Mux_2 have same select input S2. If S2 is at logic 1, the data path through module Round_O will be selected otherwise data path is through module Round_E. Each round consist of Mix and Permutation operations as described in section 3.2.1 and Figure 3.1. Complete Hardware architecture using sequential technique is depicted in Figure 7.3.

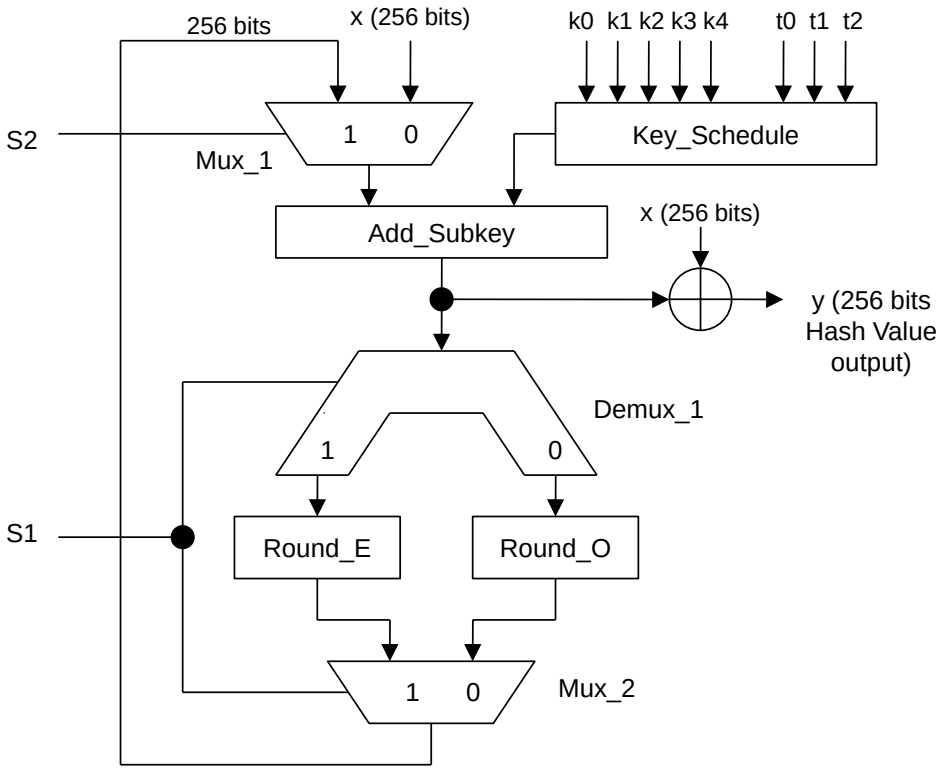


Figure 7.3: Hardware Implementation using Sequential Design

7.2.3 Threefish Rounds

Round_E and Round_O modules are same, except the value of left shift constant R involved in Mix operation. Modules Round_E and Round_O are explained in Figure 7.4.

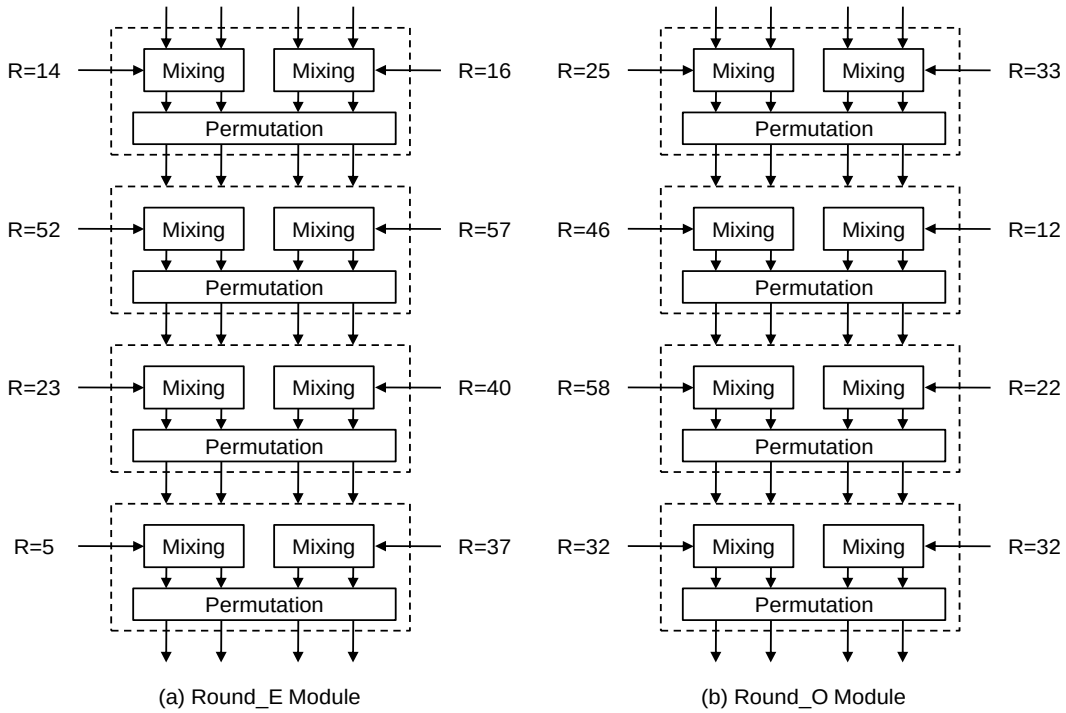


Figure 7.4: Details of Round_E and Round_O Modules

7.2.4 Key Schedule Module

Hardware architecture of key schedule module is shown in Figure 7.5. A preprocessing circuit as explained in section 6.2 generates and provides extended key(k4) and extended tweak(t2) to key schedule module. We had supposed that the two parameters k4 and t2 are available at the start of the circuit operation and are loaded into the circular shift registers k(320 bits) and t(192 bits).

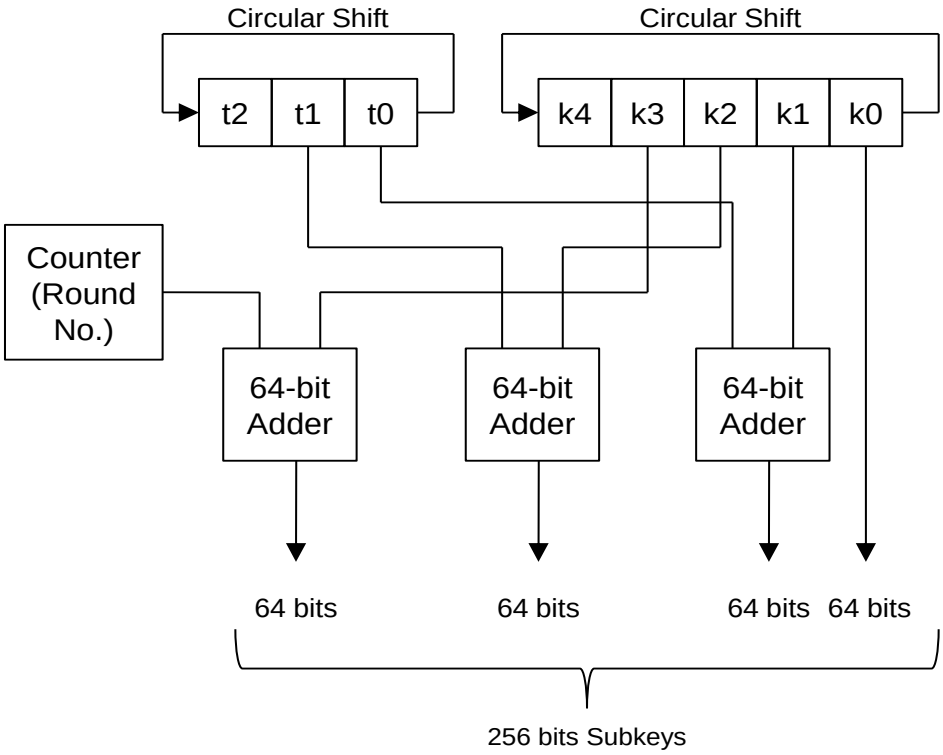


Figure 7.5: Key Schedule Module

Key Schedule module generates subkeys on every falling edge of clock on the basis of initial key(k0,k1,k2,k3) and tweak value(t0,t1). Add_Subkey, Round_O, and Round_E modules give output on the rising edge of each clock pulse. Next subkey is available on falling edge of the same clock. In this way one clock cycle is required to complete four rounds, subkey addition and subkey generation. Therefore to complete 72 rounds and 19 Subkey addition of Skein-256, 19 clocks will be required. Final hash value will be available after 19 clock cycles at the output of the XOR gate (which is to create the UBI construction of the algorithm).

Chapter 8

RESULTS

In this chapter, we reported our results and compared these with other available implementations in terms of area, throughput and throughput per area.

8.1 Achieved Results

For synthesis targeting the Xilinx FPGAs, the ISE tools (v9.2) have been used. The target devices were a Xilinx Spartan 3 5000, speed grade 5, package FG900 (xc3s5000-5fg900) and a Xilinx Virtex 5 LX100, speed grade 3, package FF1760 (xc5vlx110-3ff1760). For both FPGA architecture, the device resource usage in terms of number of slices and clock frequency estimation after synthesis are reported. For Spartan 3, a slice consists of two 4-input LUTs and two flip flops [1], while for Virtex 5, a CLB slice contains four 6-input look-up tables (LUTs) and four configurable flipflops [18]. No other functional blocks (e.g., Block RAMs) have been used by our implementations.

Table 8.1 shows the result of combinational design while Table 8.2 shows the results of design with sequential architecture.

Table 8.1: Synthesis Results of Combinational Design for Skein-256

FPGA	**Area** [Slices]	**Clock Freq.** [MHz]	**Time Delay(t)** [nSec]
Xilinx Spartan 3	8239	3.539	282.537
Xilinx Virtex 5	3658	8.531	117.214

Table 8.2: Synthesis Results of Sequential Design for Skein-256

FPGA	**Area** [Slices]	**Clock Freq.** [MHz]	**Time Delay(t)** [nSec]
Xilinx Spartan 3	2317	43.530	23.209
Xilinx Virtex 5	821	119.712	8.353

8.2 Throughput

From the results given in previous section, we can calculate throughput of design. The throughput of a given design can be calculated by

$$Throughput(T.P) = \frac{BlockSize}{T} \tag{8.1}$$

where BlockSize is the block size of message in bits, which is 256 bits for Skein-256. T is the total time required to calculate hash value which is given by

$$T = TimeDelay(t) \cdot Required\_Clock\_Cycles \tag{8.2}$$

Therefore throughput of combinational design is shown in Table 8.3

Table 8.3: Throughput of Combinational Design for Skein-256

FPGA	**Area** [Slices]	**Time Delay(t)** [nSec]	**Throughput** = 256 / t [Gbits/sec]
Xilinx Spartan 3	8239	282.537	0.906
Xilinx Virtex 5	3658	117.214	2.18

As mentioned in section 7.2 that 19 clock cycles are required to calculate the final hash value. Throughput of sequential designs is shown in Table 8.4.

8.3 Comparison with previous work

Table 8.5 shows the comparison of results with available implementations in terms of throughput.

Table 8.6 shows the comparison of results with available implementations in terms of area.

Table 8.4: Throughput of Sequential Design for Skein-256

FPGA	**Area** [Slices]	**Time Delay(t)** [nSec]	**Total Time Delay** T = t X 19 [nSec]	**T.P** = 256 / T [Gbits/sec]
Xilinx Spartan 3	2317	23.209	440.971	0.580
Xilinx Virtex 5	821	8.353	158.707	1.61

Table 8.5: Throughput Comparison of Skein-256

FPGA	**OurDesign** [Gbps]		**[15]** [Gbps]	**[16]** [Gbps]	**[19]** [Gbps]	**[20]** [Gbps]	**[21]** [Gbps]
	Comb	Sequ					
Xilinx Spartan 3	**0.906**	**0.580**	0.669	-	-	-	-
Xilinx Virtex 5	**2.18**	**1.61**	1.751	0.409	1.567	0.973	1.402

Table 8.6: Area Comparison of Skein-256

FPGA	**OurDesign** [Slices]		**[15]** [Slices]	**[16]** [Slices]	**[19]** [Slices]	**[20]** [Slices]	**[21]** [Slices]
	Comb	Sequ					
Xilinx Spartan 3	**8239**	**2317**	2421	-	-	-	-
Xilinx Virtex 5	**3658**	**821**	937	932	843	893	854

Then we compare throughput per area of implementations. Throughput per area is an important parameter to compare the implementation results. Table 8.7 shows the comparison of results with available implementations in terms of throughput/area.

Table 8.7: ThrouArea Comparison of Skein-256

FPGA	**OurDesign** Mbps/Slices		**[15]** Mbps/ Slices	**[16]** Mbps/ Slices	**[19]** [Mbps/ Slices	**[20]** Mbps/ Slices	**[21]** Mbps/ Slices
	Comb	Sequ					
Xilinx Spartan 3	**8239**	**2317**	2421	-	-	-	-
Xilinx Virtex 5	**3658**	**821**	937	932	843	893	854

Chapter 9

CONCLUSION

9.1 Concluding Remarks

In this work we presented the design of our high-speed hardware implementation of the hash function Skein-256. We reported the performance figures of our implementation in terms of throughput, area and throughput/area in addition we compare our results with available results. Achieved results in this work are meeting/exceeding to the various well known implementations in past years.

9.2 Future Work

We used the basic variant i.e. Skein-256 for our implementation. Other variants of Skein are Skein-512 and Skein-1024. Present work may easily be modified for Skein-512 because it contains the same number of rounds. However, Skein-1024 requires 80 rounds rather than 72 rounds.

We chose Skein out of the 14 candidates which advanced to round 2. Now NIST has announced the five short listed candidates for the round 3 of the competition. One or more candidates other than the Skein may be chosen for future work.

Furthermore, pipelining the design at appropriate points may result in very high throughput rates. Existing design may be enhanced by using pipelining techniques.

Bibliography

[1] Xilinx, Inc., "Spartan-3E FPGA family: Complete datasheet," *http://www.xilinx.com/support/documentation/datasheets/ds312.pdf*, April 2008.

[2] National Institute of Standards and Technology (NIST), "Cryptographic Hash Algorithm Competition," *http://www.nist.gov/itl/csd/ct/*.

[3] A. H. Namin and M. A. Hasan, "Hardware Implementation of the Compression Function for Selected SHA-3 Candidates," pp. 1–29.

[4] M.Liskov, R.Rivest, D. Wagner, "Tweakable Block Ciphers," *Springer-Verlag*, pp. 31–46, 2002.

[5] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu, "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD," *Cryptology eprint Archive, Report 2004/199, http://eprint.iacr.org/2004/199*, 2004.

[6] Marc Stevens, "Fast Collision Attack on MD5," *eprint-2006-104-S*.

[7] Michael Szydlo, "SHA-1 Collisions can be Found in 2^{63} Operations," *CryptoBytes Technical Newsletter*, August 19, 2005.

[8] P. van Oorschot and M. Wiener, "Parallel collision search with application to hash functions and discrete logarithms," *Proceedings of 2nd ACM Conference on Computer and Communication Security*, 1994.

[9] National Institute of Standards and Technology (NIST), "Federal Register Notices/ Vol. 72, No. 212 / Friday, November 2, 2007," *http://www.csrc.nist.gov/groups/ST/hash/documents/*.

[10] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, Jesse Walker, "The Skein Hash Function Family Version 1.2," *http://www.schneier.com/skein.html*, pp. 1–84, Sep 2009.

[11] "The Field Programmable Gate Array (FPGA): Expanding Its Boundries," *Instant Market Research*, April 2006.

[12] National Instruments, "Introduction to FPGA Technology: Top Five Benefits," *http://zone.ni.com/devzone/cda/tut/p/id/6984*, June 2008.

[13] BDTI, "FPGAs for DSP," *BDTI Focus Report*, vol. BDTI Benchmarking, 2006.

[14] M. Thompson, "FPGAs accelerate time to market for industrial designs," *EE Times*, July 2004.

[15] Stefan Tillich, "Hardware Implementation of the SHA-3 Candidate Skein," *http://www.eprint.iacr.org/2009/159.pdf*, pp. 1–7.

[16] Men Long, "Implementing Skein Hash Function on Xilinx Virtex-5 FPGA Platform," *http://www.schneier.com/skein_fpga.pdf*, pp. 1–15.

[17] P. O. A. Menezes and S. Vantone, *HandBook of Applied Cryptograghy*. The CRC Press series on discrete mathematics and its applications, 1996.

[18] Xilinx, Inc., "Virtex-5 FPGA User Guide," *http://www.xilinx.com*, March 2009.

[19] Kris Gaj, Ekawat Homsirikamol, and Marcin Rogawski, "Fair and Comprehensive Methodology for Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates Using FPGAs," *eprint.iacr.org/2010/445.pdf*, 2010.

[20] B. Baldwin, N. Hanley, M. Hamilton, L. Lu, A. Byrne, M. ONeill and W. P.Marnane, "FPGA Implementations of the Round Two SHA-3 Candidates," *Second SHA-3 Candidate Conference August 23-24 2010*.

[21] S. Matsuo, M. Knezevic, P. Schaumont, I. Verbauwhede, A. Satoh, K. Sakiyama and K. Ota, "How Can We Conduct Fair and Consistent Hardware Evaluation for SHA-3 Candidate?," *Second SHA-3 candidate conference August 23-24 2010*.