Secure and Efficient Implementations of Cryptographic Primitives

Xu Guo

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Patrick Schaumont, Chair
Ezra Brown
Cameron Patterson
Sandeep Shukla
Joe Tront

May 1, 2012
Blacksburg, Virginia

Keywords: System-on-Chip, Cryptographic Coprocessor, Elliptic Curve Cryptography, Hash Function, Block Cipher, SHA-3, Side-Channel Attacks, Fault Attacks

Copyright 2012, Xu Guo

Secure and Efficient Implementations of Cryptographic Primitives

Xu Guo

Abstract

Nowadays pervasive computing opens up many new challenges. Personal and sensitive data and computations are distributed over a wide range of computing devices. This presents great challenges in cryptographic system designs: how to protect privacy, authentication, and integrity in this distributed and connected computing world, and how to satisfy the requirements of different platforms, ranging from resource constrained embedded devices to high-end servers. Moreover, once mathematically strong cryptographic algorithms are implemented in either software or hardware, they are known to be vulnerable to various implementation attacks. Although many countermeasures have been proposed, selecting and integrating a set of countermeasures thwarting multiple attacks into a single design is far from trivial. Security, performance and cost need to be considered together.

The research presented in this dissertation deals with the secure and efficient implementation of cryptographic primitives. We focus on how to integrate cryptographic coprocessors in an efficient and secure way. The outcome of this research leads to four contributions to hardware security research. First, we propose a programmable and parallel Elliptic Curve Cryptography (ECC) coprocessor architecture. We use a systematic way of analyzing the impact of System-on-Chip (SoC) integration to the cryptographic coprocessor performance and optimize the hardware/software codesign of cryptographic coprocessors. Second, we provide a hardware evaluation methodology to the NIST SHA-3 standardization process. Our research efforts cover both of the SHA-3 fourteen Second Round candidates and five Third Round finalists. We design the first SHA-3 benchmark chip and discuss the technology impact to the SHA-3 hardware evaluation process. Third, we discuss two technology dependent issues in the fair comparison of cryptographic hardware. We provide a systematic approach to do a cross-platform comparison between SHA-3 FPGA and ASIC benchmarking results and propose a methodology for lightweight hash designs. Finally, we provide guidelines to select implementation attack countermeasures in ECC cryptosystem designs. We discuss how to integrate a set of countermeasures to resist a collection of side-channel analysis (SCA) attacks and fault attacks.

The first part of the dissertation discusses how system integration can affect the efficiency of the cryptographic primitives. We focus on the SoC integration of cryptographic coproces-

sors and analyze the system profile in a co-simulation environment and then on an actual FPGA-based SoC platform. We use this system-level design flow to analyze the SoC integration issues of two block ciphers: the existing Advanced Encryption Standard (AES) and a newly proposed lightweight cipher PRESENT. Next, we use hardware/software codesign techniques to design a programmable ECC coprocessor architecture which is highly flexible and scalable for system integration into a SoC architecture.

The second part of the dissertation describes our efforts in designing a hardware evaluation methodology applied to the NIST SHA-3 standardization process. Our Application Specific Integrated Circuit (ASIC) implementation results of five SHA-3 finalists are the ***first*** ASIC real measurement results reported in the literature. As a contribution to the NIST SHA-3 competition, we provide timely ASIC implementation cost and performance results of the five SHA-3 finalists in the SHA-3 standard final round evaluation process. We define a consistent and comprehensive hardware evaluation methodology to the NIST SHA-3 standardization process from Field Programmable Gate Array (FPGA) prototyping to ASIC implementation.

The third part of the dissertation extends the discussion on hardware benchmarking of NIST SHA-3 candidates by analyzing the impact of technology to the fair comparison of cryptographic hardware. First, a cross-platform comparison between the FPGA and ASIC results of SHA-3 designs demonstrates the gap between two sets of benchmarking results. We describe a systematic approach to analyze a SHA-3 hardware benchmark process for both FPGAs and ASICs. Next, by observing the interaction of hash algorithm design, architecture design, and technology mapping, we propose a methodology for lightweight hash implementation and apply it to CubeHash optimizations. Our ultra-lightweight design of the CubeHash algorithm represents the ***smallest*** ASIC implementation of this algorithm reported in the literature. Then, we introduced a cost model for analyzing the hardware cost of lightweight hash implementations.

The fourth part of the dissertation discusses SCA attacks and fault attacks resistant cryptosystem designs. We complete a comprehensive survey of state-of-the-art of secure ECC implementations and propose a methodology on selecting countermeasures to thwart multiple side-channel attacks and fault attacks. We focus on a systematic way of organizing and understanding known attacks and countermeasures.

Acknowledgments

Writing this personal letter of gratitude makes me recall so many nice people and all the wonderful moments. I was very fortunate to work with many worldwide excellent researchers and make a lot of good friends during the last five years. Several individuals deserve my greatest gratitude for their guidance and support in my PhD research and life.

First of all, I want to express my greatest appreciation to my PhD advisor, Prof. Patrick Schaumont, who provides me with persistent guidance and so many opportunities to expand myself not only academically but personally as well. He sets an excellent model for me as one of the hardest working and dedicated researchers with great passion.

I am honored to have Prof. Sandeep Shukla, Prof. Ezra Brown, Prof. Cameron Patterson and Prof. Joe Tront as my PhD advisory committee. Thank you for many valuable suggestions and helpful discussions in making my research plan and finalizing the dissertation.

I would like to extend my greatest appreciation to many worldwide research collaborators. I learned a lot from Dr. Junfeng Fan, Dr. Elke De Mulder, Dr. Miroslav Knežević and Prof. Ingrid Verbauwhede from COSIC at K.U.Leuven, and had a great time to work together to make some good progress. I also learned how to play an active role in some large internationally collaborated project, and hence I would like to acknowledge those co-authors and people who have inspired my research: Prof. Daniel Bernstein, Prof. Jens-Peter Kaps, Prof. Kris Gaj, Prof. Kazuo Sakiyama, Dr. Akashi Satoh, Dr. Shinichiro Matsuo, Dr. Toshihiro Katashita, and Kazuyuki Kobayashi. I am also very grateful to several CESCA

faculties and students who helped me a lot through inter-group research collaborations: Prof. Leyla Nazhandali, Prof. Michael Hsiao, Dr. Michael B. Henry, Meeta Srivistav, Sinan Huang, Dinesh Ganta, and Yongbo Zuo. A special thanks goes to all my colleagues in the SES group, including Dr. Zhimin Chen, Dr. Abhranil Maiti, Michael Gora, Sergey Morozov, Anand Reddy, Raghunandan Nagesh, Ambuj Sinha, Srikrishna Iyer, Christian Tergino, Francisco Borelli, Suvarna Mane, Lyndon Judge, Michael Cantrell, and Mostafa Taha, who made my working environment and life in Blacksburg very pleasant.

I am very grateful to all of my friends in Blacksburg who made my oversea life feel like staying at home. Rather than trying to be exhaustive and end up with a lengthy enumeration of names, I would like to remember everyone of you by heart and simply say: *my life is so great with all of you*.

Finally, I would like to thank my parents, my wife, and my big family. Without their encouragement and love, it would have been impossible for me to complete the dissertation. I hope that I can continue to make them as proud of me as I am of them.

Contents

List of Figures

List of Tables

List of Acronyms

AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
API	Application Program Interface
ASIC	Application Specific Integrated Circuit
CC	Cycle Count
CHES	Cryptographic Hardware and Embedded Systems
CMOS	Complementary Metal-Oxide-Semiconductor
CPA	Correlation Power Analysis
CPU	Central Processing Unite
DEMA	Differential Electromagnetic Analysis
DES	Data Encryption Standard
3DES	Triple Data Encryption Standard
DFA	Differential Fault Analysis
DFF	D Flip-Flop
DPA	Differential Power Analysis
DRC	Design Rule Check
DSA	Digital Signature Algorithm
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Logarithm Problem

ECDSA	Elliptic Curve Digital Signature Algorithm
ECPM	Elliptic Curve Point Multiplication
ECSM	Elliptic Curve Scalar Multiplication
EM	Electromagnetic
EMA	Electromagnetic Analysis
FA	Fault Analysis
FIFO	Fist-In-First-Out
FIPS	Federal Information Processing Standard
FPGA	Field-Programmable Gate Array
FSL	Fast Simplex Link
FSM	Finite State Machine
FSMD	Finite State Machine with Datapath
GE	Gate Equivalent
HD	Hard Disk
HDL	Hardware Description Language
HMAC	Hash-based Message Authentication Codes
HW/SW	Hardware/Software
IP	Intellectual Property
ISO	International Organization for Standardization
ISS	Instruction-Set Simulator
MB	MicroBlaze
MEM	Memory
MMH	Multiple-Message Hash
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OPB	On-chip Peripheral Bus
PA	Power Analysis
PB	PicoBlaze

PCB	Printed Circuit Board
PKC	Public-Key Cryptography
PLB	Processor Local Bus
PV	Point Verification
RFID	Radio Frequency Identification
RSA	Rivest-Shamir-Adleman Algorithm
RTL	Register Transfer Level
SCA	Side-Channel Analysis
SDF	Standard Delay Format
SEMA	Simple Electromagnetic Analysis
SHA	Secure Hash Algorithm
SMH	Single-Message Hash
SoC	System-on-Chip
SPA	Simple Power Analysis
TC	Toggle Count
XPS	Xilinx Platform Studio

Chapter 1

Introduction

The rapid growth of electronic communication implies that issues in information security are of increasing practical importance. Nowadays pervasive computing opens up many new challenges. Personal data and sensitive data and computations are distributed over a wide range of computing devices, connected together through wired and wireless links. Some novel applications also impose new security requirements: the Radio Frequency Identification (RFID) tag that is going to replace traditional bar codes requires extremely lightweight cryptography; portable devices (e.g. smart phones) that are connected to Internet need various cryptography services that are low power, fast and secure; Internet servers require ultra high performance and parallel cryptographic implementations. All of these present great challenges in security issues and protocols: how to protect privacy, authentication, and integrity in this distributed and connected computing world; how to satisfy the requirements of different platforms, ranging from resource constrained embedded devices to high-end servers. One key to this is secure and efficient implementations of the cryptographic primitives that support these security protocols. The research described in this dissertation tries to identify several research problems in cryptographic primitive designs, especially from a system integration point of view, and it provides solutions to address them.

1.1 Challenges with Cryptographic Primitive Implementations

Cryptographic primitives serve as the basic building blocks for creating complex cryptographic systems. Since the last decade cryptography primitives have moved on with increasing security demands. For private-key cryptography, the Advanced Encryption Standard (AES) has replaced Triple Data Encryption Standard (3DES). For public-key cryptography, due to a number of influential discoveries, Elliptic Curve Cryptography (ECC) is now considered an ideal substitute for the early RSA standard in many applications. For one-way hash function, National Institute of Standards and Technology (NIST) is hosting an ongoing SHA-3 competition, and the winner will substitute the old NIST SHA-2 family of hash functions (i.e., SHA-224, SHA-256, SHA-384 and SHA-512) as the new hash function standard by 2012. All these cryptographic primitives should be implemented in a highly reliable and efficient way. Furthermore, the efficiency and security of the cryptographic primitives must be re-evaluated after system integration on various platforms.

Two problems rise up when security engineers face the current challenging situation of a large diversity of platforms, novel cryptographic applications, and newly established cryptographic standards:

First, how to do system integration of cryptographic primitives in a secure and efficient way? The goal of system integration of cryptographic primitives is always trying to find a tradeoff between cost, performance and security. Design for flexibility is also important as it is always better to provide a scalable solution that is capable to cover a range of applications and platforms with different requirements and constraints. Furthermore, how to implement countermeasures to resist multiple attacks of different types is far from trivial. The attacks under consideration are the implementation attacks (e.g. Side-Channel Analysis (SCA) attacks) that do not exploit inherent weakness of an algorithm (like classical cryptanalysis), but rather a characteristic of the implementation. Compared to the brute force

attack on AES which requires 10^{13} years in theory, SCA attacks can break an unprotected embedded software implementation of AES within minutes.

Second, how to develop a fair and comprehensive methodology to evaluate and compare the performance of competing cryptographic algorithms? Cryptographic primitive implementations can be considered as Intellectual Properties (IPs) that are ready for system integration, and hence the results of different implementations of the same primitive should be comparable. However, in the average crypto-hardware conference proceedings, one will find that no two authors measure resource cost or performance of hardware implementations using the same metrics. For example, the 11 tables that compare hardware implementations in the proceedings of CHES 2008 contain 18 different metrics for hardware cost and 10 different metrics for hardware performance. While one author may use clock cycles, another one may use nanoseconds, and a third one blocks-per-second. As for a practical need, the solution to solve this problem can also benefit the ongoing NIST SHA-3 competition when comparing the performance of different SHA-3 candidates.

The research presented in this dissertation is to explore efficient and secure architectures and design strategies for system integration of cryptographic primitives for SoC platforms.

1.2 Dissertation Organization and Contributions

This section gives an overview of the structure of the dissertation and highlights the personal contributions.

Chapter 1: Introduction. The first chapter starts with identifying the challenges with cryptographic primitive implementations and motivates our research topics. We also present a brief summary and contributions of each research topic.

Chapter 2: Background. The scope of this research covers several aspects of cryptology research. This chapter positions the topic in the broad field of cryptography. It give an

overview of related cryptographic algorithms and protocols.

Chapter 3: Hardware/Software Codesign for Cryptographic Coprocessors. This chapter presents a system-level design flow, covering simulation up to FPGA implementation, that evaluates the performance and power consumption of a cryptographic coprocessor integrated in a complete system. After a detailed analysis of hardware/software (HW/SW) codesign of cryptographic coprocessors, we demonstrate that the SoC may become performance-limited due to the coprocessor data- and instruction-transfers. This research presents several contributions to codesigns for cryptosystems: (1) to identify that beyond performance consideration the bus selection can also be a tradeoff between power efficiency and energy efficiency; (2) to point out that a lightweight and power-efficient cipher (PRESENT) integrated in a SoC environment may actually be less energy-efficient than a standard block cipher (AES); (3) to present a complete ECC SoC design and focus on the system integration issues on a real FPGA platform; (4) to quantify the impact of control hierarchy and local storage in the coprocessor, and show how the system performance regions are affected; (5) to optimize the ECC coprocessor to have high flexibility by proposing a novel parallel architecture, which can be used to explore application- and algorithm-level parallelism of ECC.

Chapter 4: Hardware Evaluation of SHA-3 Candidates. This chapter takes the NIST SHA-3 competition as a practical case study, and we propose a consistent and systematic approach to move a SHA-3 hardware benchmark process from FPGA prototyping to ASIC implementation. Comprehensive evaluation results on fourteen Second Round SHA-3 candidates and five Third Round SHA-3 finalists are provided. This research contributes to the NIST SHA-3 hardware evaluation process by proposing a consistent SHA-3 hardware benchmark methodology and providing timely cost and performance results on the ***first*** SHA-3 ASIC. This chip is fabricated in 0.13 μm IBM process using standard cell Complementary Metal-Oxide-Semiconductor (CMOS) technology. We provide measurement results of all the five finalists using the latest Round 3 tweaks.

Chapter 5: The Impact of Technology in Fair Comparison of Cryptographic Hardware. In this chapter, we extend the SHA-3 hardware evaluation work by discussing two technology dependent issues which may significantly affect cryptographic hardware evaluation results. We first discuss how to compare FPGA and ASIC results with fixed HDL designs and provides some insights in how to look at the SHA-3 FPGA and ASIC benchmarking results together. In addition to the fair comparison between different SHA-3 hardware implementations using common metrics, another important evaluation criterion is the ability of the candidate algorithm to be implemented on resource-constrained platforms. By observing the interaction of hash algorithm design, architecture design, and technology mapping, we propose a methodology for lightweight hash implementations and apply it to CubeHash (SHA-3 Second Round Candidate) optimizations. Our ultra-lightweight design of CubeHash represents the ***smallest*** ASIC implementation of this algorithm reported in the literature.

Chapter 6: Secure Implementations of ECC Cryptosystems. Cryptographic implementations are known to be vulnerable to various SCA attacks (such as Power Analysis (PA) [94] attacks and Electromagnetic Analysis (EMA) attacks [1]) and fault attacks [25]. When considering secure HW/SW codesign of cryptographic primitives, not only the coprocessor but also the system-level protection must be taken into account (e.g. protect the communication channel between hardware and software) since a cryptosystem will fail at its weakest link [134]. This work contributes to the security analysis of cryptosystem by presenting a study of state-of-the-art of SCA attacks and fault attacks on ECC and provide several guidelines in how to select countermeasures to resist multiple attacks in a single design.

Chapter 7: Conclusions and Future Work. In this final chapter, conclusions are drawn and some future work is discussed.

Chapter 2

Background

The scope of this research covers several aspects of cryptology research. This chapter gives an overview of related cryptographic algorithms and protocols, and wraps up with a definition of the research space in this dissertation.

2.1 Overview of Cryptology

Cryptology is a more general term than cryptography, which can be split into two main branches as shown in Fig. 2.1:

Cryptography is the science of secret writing with the goal of hiding the meaning of a message and supporting information security. It includes Private Key Cryptography, Public Key Cryptography, Hash Functions and Cryptographic Protocols.

Cryptanalysis is the study of methods of obtaining the meaning of encrypted information by exploring the weakness of the cryptography. The algorithm cryptanalysis falls into two categories: analytical attacks, which exploit the internal structure of the encryption method, and brute-force attacks, which treat the encryption algorithm as a black box and search all the possible keys. The more powerful modern attacks are implementation attacks,

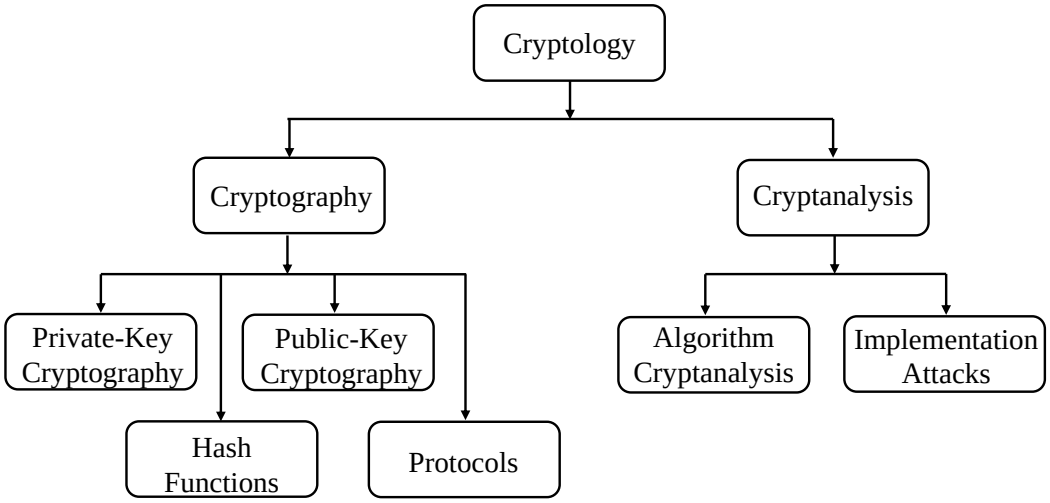


Figure 2.1: An overview the research field of cryptology

such as SCA, EMA and FA attacks.

2.2 Private-Key Cryptography

Private-key ciphers or symmetric ciphers are a class of algorithms for cryptography that use trivially related, often identical, cryptographic keys for both decryption and encryption. The symmetric ciphers can be classified into stream ciphers and block ciphers. Stream ciphers encrypt the bits of the message one bit at a time, and the block ciphers take a number of bits and encryption them as single unit. Rijndael has been selected by NIST as the AES after a critical assessment, which included extensive benchmarking on a variety of platforms ranging from smart cards [71] to high end parallel machines [139]. However, still many new block ciphers were proposed with special implementation properties, such as TEA [138], IDEA [100], Hight [78], Clefia [126], DESXL [102], and PRESENT [44].

2.2.1 PRESENT Block Cipher

In this research, we are especially interested in comparing AES with the newly published PRESENT block cipher, which was designed with area and power constraints in mind.

PRESENT is an SPN-based (substitution permutation network) block cipher with 31 rounds, a block size of 64-bit, and a key size of 80-bit or 128-bit. Fig. 2.2 shows the top level algorithmic description and hardware structure of PRESENT, which is similar with those of AES. So, PRESENT looks like a simplified version of AES with a relatively lower security level. The PRESENT comprises three stages: a key-mixing step, a substitution layer, and a permutation layer. For the key mixing, simply a XOR is used. The key schedule consists essentially of a 61-bit rotation together with an S-box and a round counter (PRESENT-80 uses a single Sbox, whereas PRESENT-128 requires two S-boxes). The substitution layer comprises 16 S-boxes with 4 input bits and 4 output bits. Similar S-boxes are used in both the data path and the key scheduling. The permutation layer is a bit transposition and can be realized by direct wiring [44].

2.3 Public-Key Cryptography

Public-key cryptography is used for a secure distribution of secret keys and some important forms of authentication and non-repudiation (e.g. digital signatures). Curve-based cryptography (e.g. ECC) has become very popular in the past several years [90]. ECC provides the same level of security as RSA with considerably shorter operands (approximately 160-256 bit vs. 1024-3072 bit). In many cases, ECC has performance advantages (fewer computations) and bandwidth advantages (shorter signatures and keys) over RSA. However, ECC is still considered as a computational intensive application due to the complexity of scalar or point multiplications. A scalar multiplication, $k \cdot P$, with k is an integer and P is a point on an elliptic curve, needs to be realized through a sequence of point additions and doublings. These group operations can be further decomposed in several types of finite field arithmetic

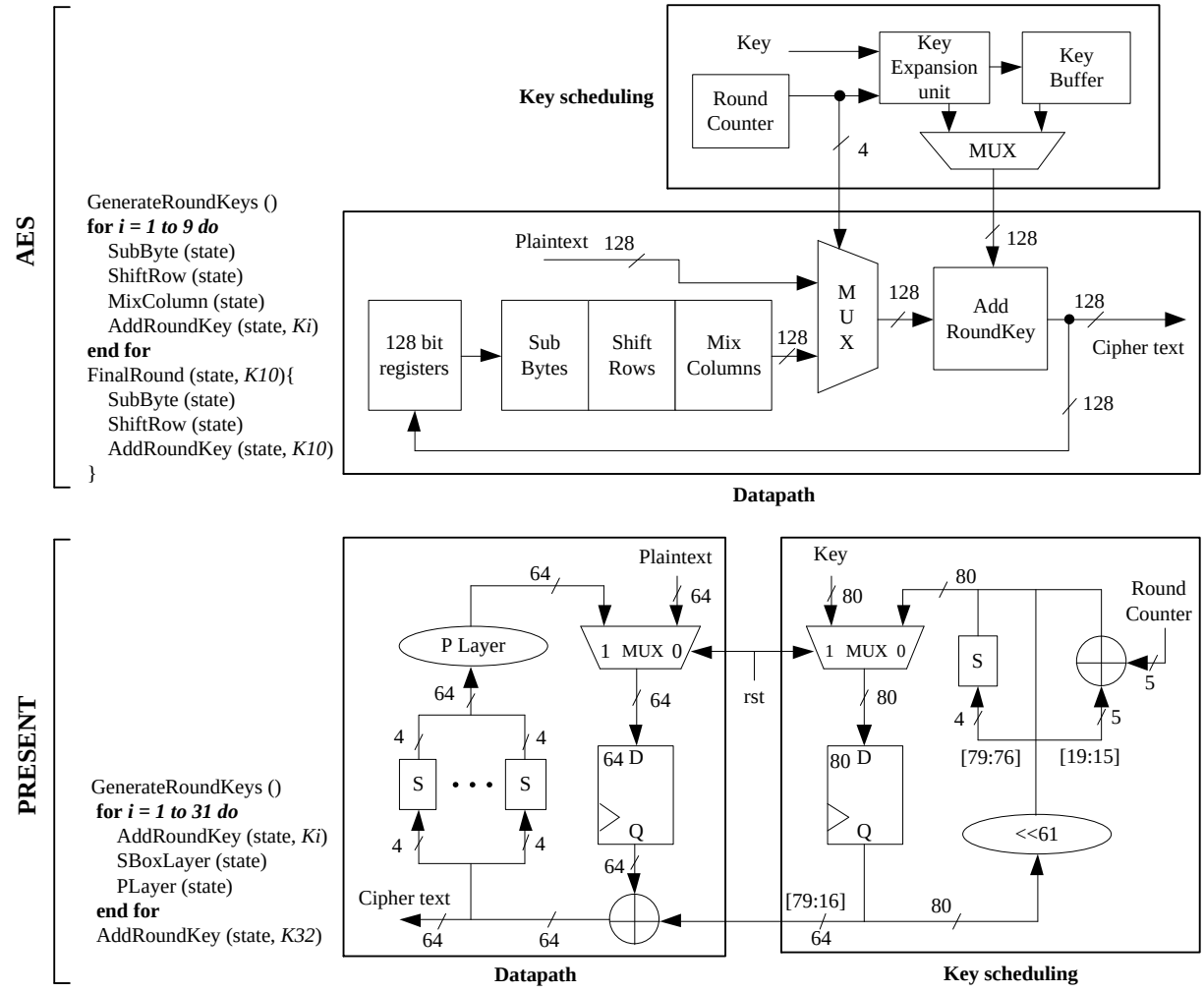


Figure 2.2: Comparison of algorithmic descriptions and hardware structures between AES-128 and PRESENT-80.

with ultra-long operand word length (e.g. 163-bit or above).

2.3.1 Elliptic Curve Cryptography

We give a brief introduction to ECC in this section. A comprehensive introduction to ECC can be found in [11, 26].

We assume the notations below are defined as follows:

- $\mathbb{K}$: a finite field;
- $char(\mathbb{K})$: the characteristic of $\mathbb{K}$;
- $E(a_1, a_2, a_3, a_4, a_6)$: an elliptic curve with coefficients a_1, a_2, a_3, a_4, a_6;
- $P(x, y)$: a point with coordinates (x, y);
- $\mathcal{O}$: point at infinity;
- $E(\mathbb{K})$: a group formed by the points on an elliptic curve E defined over the finite field $\mathbb{K}$;
- $\#E$: the number of points on the curve E, i.e. the order of the curve E;
- *weak* curve: a curve whose order does not have big prime divisors;
- the order of point P: the smallest integer r such that $r \cdot P = \mathcal{O}$;
- coordinate system: a system to represent a point in an n-dimensional space;
- affine coordinates: a point is represented with a two-tuple of numbers (x, y);
- projective coordinates: a point (x, y) is represented as (X, Y, Z), where $x = X/Z, y = Y/Z$;
- Jacobian projective coordinates: a point (x, y) is represented as (X, Y, Z), where $x = X/Z^2, y = Y/Z^3$.

Elliptic Curve Cryptosystems

An elliptic curve E over a field $\mathbb{K}$ can be defined by a *Weierstrass* equation.

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.1}$$

where $a_1, a_2, a_3, a_4, a_6 \in \mathbb{K}$ and $\Delta \neq 0$. Here Δ is the discriminant of E. A *Weierstrass* equation can be simplified by applying change of coordinates. If $char(\mathbb{K})$ is not equal to 2 or 3, then E can be transformed to

$$y^2 = x^3 + ax + b \tag{2.2}$$

where $a, b \in \mathbb{K}$. If $char(\mathbb{K}) = 2$, then E can be transformed to

$$y^2 + xy = x^3 + ax^2 + b \tag{2.3}$$

if E is non-supersingular.

For cryptographic use, we are only interested in elliptic curves over a finite field. Elliptic curves defined over both prime fields and binary extension fields are used in reality. Given two points, $P(x_1, y_1)$ and $Q(x_2, y_2)$, the sum of P and Q is again a point on the same curve under the addition rule. The set of points (x, y) on E together with the point at infinity form an abelian group. The security of ECC is based on the hardness of the ECDLP, namely, finding out k for two given points P and Q such that $Q = k \cdot P$. The variable k is called the scalar. In most cases the secrecy of k is protected.

Scalar Multiplication

A cryptographic device for ECC is supposed to perform scalar multiplication efficiently and securely.

Given the a point $P \in E(\mathbb{K})$ and a scalar k, the computation $k \cdot P$ is called point multiplication or scalar multiplication. Algorithm 1 shows the Left-To-Right binary method for scalar multiplication.

Algorithm 1 Left-To-Right (downwards) binary method for point multiplication

Input: $P \in E(\mathbb{F})$ and integer $k = \sum_{i=0}^{l-1} k_i 2^i$.
Output: kP.

1: $R \leftarrow \mathcal{O}$.
2: **for** $i = l - 1$ downto 0 **do**
3: $\quad R \leftarrow 2R$.
4: $\quad$ If $k_i = 1$ then $R \leftarrow R + P$.
5: **end for**

Return R.

Given two points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ on an elliptic curve E defined over binary extension fields, one can compute $P_3(x_3, y_3) = P_1 + P_2$ as follows:

$$\begin{aligned} x_3 &= \lambda^2 + a_1\lambda - a_2 - x_1 - x_2 \\ y_3 &= -y_1 - (x_3 - x_1)\lambda - a_1 x_3 - a_3 \end{aligned}$$

where

$$\lambda = \begin{cases} \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} & (x_1, y_1) = (x_2, y_2), \\ \frac{y_1 - y_2}{x_1 - x_2} & \text{otherwise.} \end{cases}$$

These formulas use coordinates in affine form, and they require the division operation. Because the coordinates are represented as finite-field elements, this division operation needs to be implemented as an finite-field inversion, a costly and complex operation.

2.4 Hash Functions

A cryptographic hash function takes arbitrary length input messages and generates a short, fixed-length digest. For a particular message, the message digest, or hash value, can be considered as the fingerprint of the message. Unlike the private-key and public-key algorithms, hash functions do not have a key. Hash functions are an essential part of many cryptographic protocols, like digital signature schemes and message authentication codes.

Depending on the source of the information, the definition or requirement for a cryptographic hashing function may be different to a certain degree; however, the most common characteristics or desired properties for a secure hashing function H with input message x are as follows:

- $H(x)$ is relatively easy for any given message x, so that the implementations of H in both hardware and software can be efficient.
- *One-Way Function*: for any given hash digest d, it is computationally impractical to find a message x such that the hash digest of x is the same as the hash digest d.
- *Weak Collision Resistance*: for any given message x, it is computationally infeasible to find another message y, such that the message y is not the same as message x but the hash digest of y and x are the same.
- *Strong Collision Resistance*: it is computationally infeasible to find two different messages, x and y such that their hash digests are the same.

SHA stands for Secure Hashing Algorithm. It is a group of hash functions published by the National Institute of Standards and Technology as a US Federal Information Processing Standard (FIPS). All of the current SHA algorithms were developed by the NSA.

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

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

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

In 2005, Wang et al. [136] was able to use a differential attack on the SHA-1 hash function to reduce the hash collision search time of more than one-hundred and thirty thousand times faster than a brute force method. Subsequent publications by other cryptanalysts were able to find hash collisions even faster. Despite the fact that those attacks cannot be extend to SHA-2, NIST decided that it was prudent to develop one or more hash functions as the transition from SHA-1 to the approved SHA-2 family is made.

2.4.1 NIST SHA-3 Competition

The SHA-3 competition organized by NIST aims to select, in three phases, a successor for the mainstream SHA-2 hash algorithms in use today. The initial information regarding the competition, its minimum requirements and its evaluation criteria, was published in the first quarter of 2007. By the end of 2008, almost two years after the initial competition was announced, all the candidates had been submitted to NIST for evaluation. 64 algorithms were submitted, among which 51 submissions qualified for the first round. By the completion of Phase I in July 2009, 14 out of the 51 hash candidate submissions were identified for further consideration as SHA-3 candidates. In December 2010, the SHA-3 competition entered into Phase III and five SHA-3 candidates were selected for further evaluation as SHA-3 finalists. The winner of the competition will be announced sometime in 2012. Throughout the NIST SHA-3 competition, in relative order of importance, NIST considered the security, cost, and algorithm and implementation characteristics of a candidate [133].

2.5 Summary

For the cryptographic primitive selection, there are various choices under each category of cryptography as shown above. In this research, for private-key cryptography, AES and PRESENT [44] block ciphers are targeted; for public-key cryptography, ECC is selected; for hash functions, all of the 14 Second Round SHA-3 candidates and 5 Third Round SHA-3 finalists are of interests. We believe most of the design methods and analysis can be generalized to other algorithms in each category since the goal of this research is to explore efficient architectures and design strategies for system integration of cryptographic primitives but not just focus on exploring special characteristics out of a particular algorithm.

Chapter 3

Hardware/Software Codesign for Cryptographic Coprocessors

This chapter focuses on the SoC integration of cryptographic coprocessors by analyzing the system profile in a cosimulation environment and on an actual FPGA-based SoC platform. The impact of system integration overhead to the system profile is analyzed. We use the HW/SW codesign of ECC as a case study to show how to address efficiency at system-level design.

3.1 System-Level Design Flow Using GEZEL

In order to narrow the gap between performance and flexibility, and to reduce the time required to complete a design, we use GEZEL to perform system-level design.

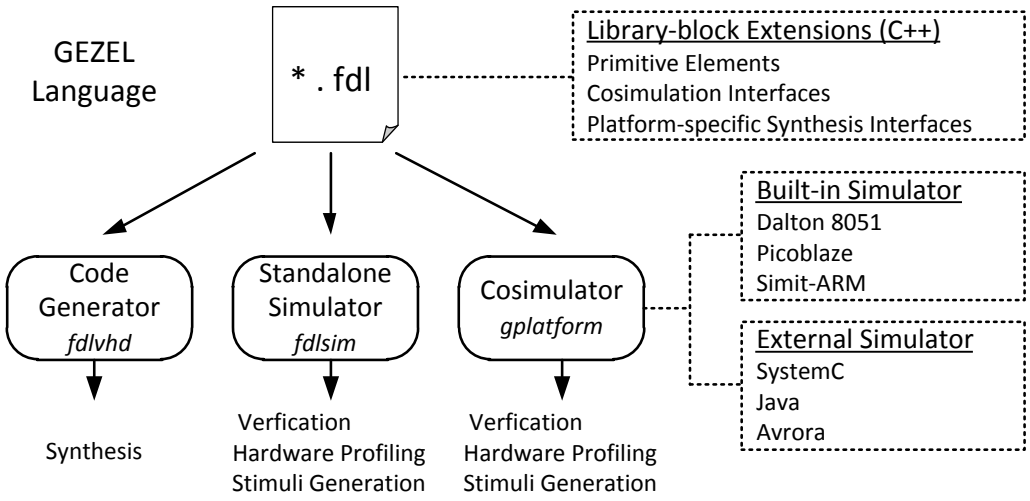


Figure 3.1: Overview of GEZEL cosimulation environment.

3.1.1 Overview

The GEZEL cosimulation environment as shown in Fig. 3.1 creates a platform simulator by combining a hardware simulation kernel with one or more ISS. The hardware part of the platform is programmed in GEZEL, a deterministic, cycle-true and implementation-oriented hardware description language. After cycle-accurate simulation, the GEZEL description of hardware can be converted into synthesizable VHDL files [122, 124]. The GEZEL cosimulation can not only verify the correctness of the coprocessor and generate the corresponding VHDLs of the function unit, but also generate the bus interface module in VHDL. Therefore, by following the GEZEL design flow, system designers do not need to make any change in hardware when doing the FPGA SoC integration of coprocessors. However, minor changes have to be made in the software driver when shifting from GEZEL cosimulation to the FPGA implementation because the ARM Instruction-Set Simulator (ISS) is replaced with the actual microprocessor (e.g. Xilinx MicroBlaze). The comparison between GEZEL cosimulation and FPGA implementation, in terms of the software driver and bus interface implementations of Processor Local Bus (PLB) and Fast Simplex Link (FSL), are illustrated in Fig. 3.2 and Fig. 3.3.

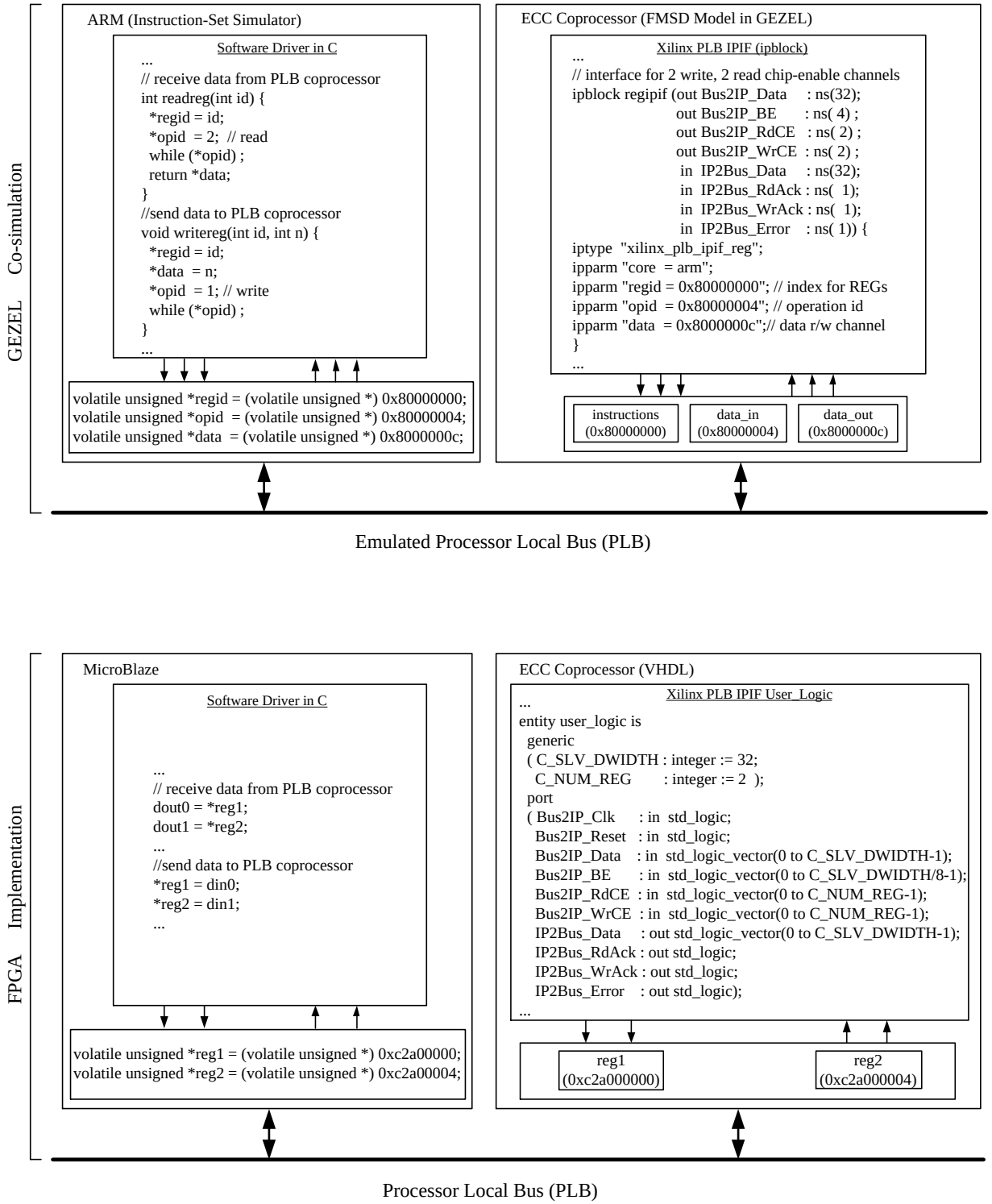


Figure 3.2: Comparison between the GEZEL cosimulation and FPGA implementation of PLB-based coprocessor.

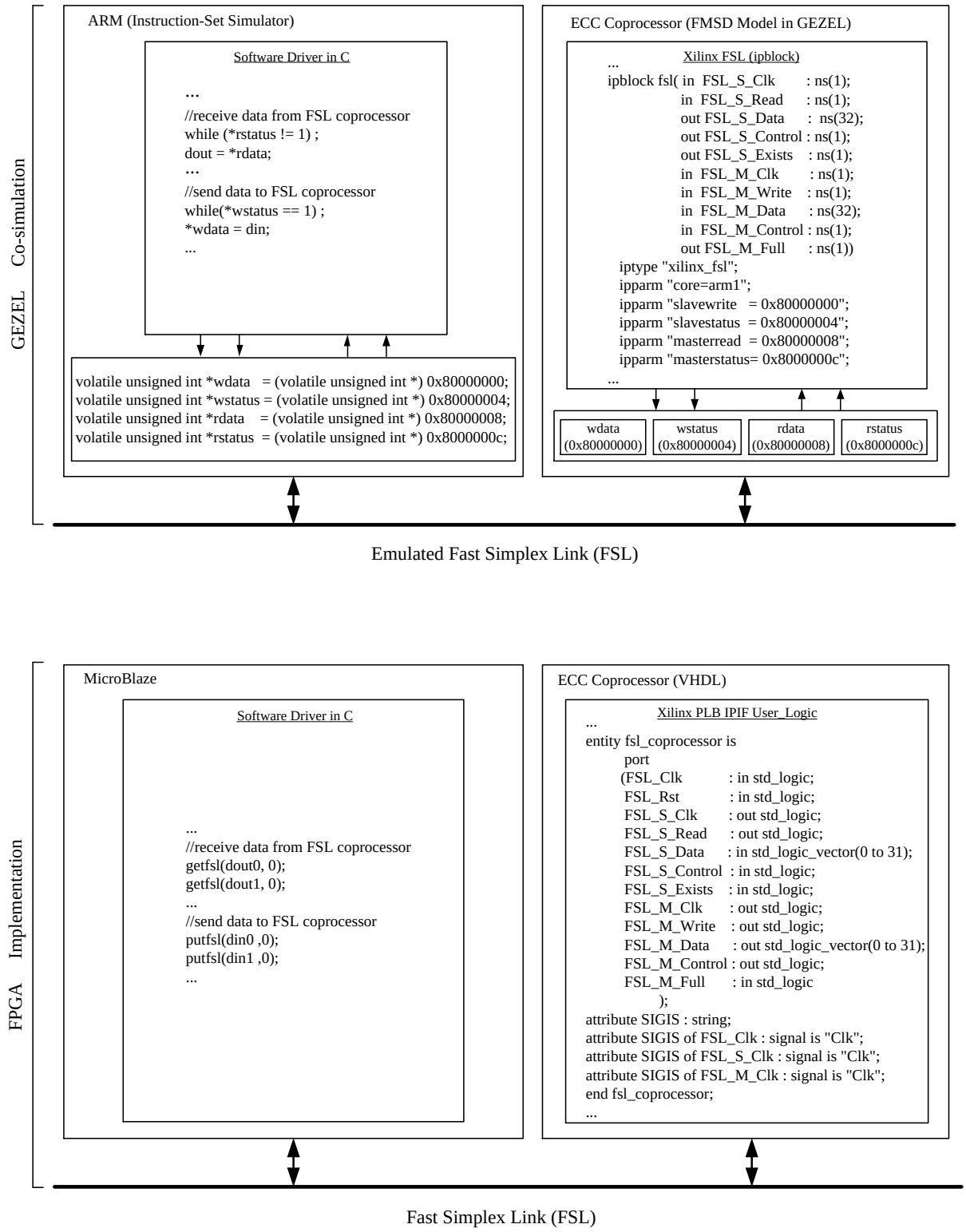


Figure 3.3: Comparison between the GEZEL cosimulation and FPGA implementation of FSL-based coprocessor.

Table 3.1: Hardware and software environment of our experiments

GEZEL Cosimulation	GEZEL Simulator	ARM ISS	arm-linux-gcc
Version	v2.3	SimIt-ARM-2.1	v3.2.1
FPGA Implementation	XPS	MicroBlaze	mb-gcc
Version	9.2.02i	7.00.b	v4.1.1
HW/SW Interfaces	PLB	FSL	
Version	v4.6	v20	

Before moving into the detailed system design and analysis under GEZEL cosimulation and FPGA implementation, we want to first specify the hardware and software environment of our experimental setup since different versions may possibly affect the experimental results presented in this work (see Table 3.1).

3.1.2 Hardware/Software Interfaces

There are three commonly available hardware/software interfaces: processor local busses (e.g. Xilinx PLBv46), direct connection busses (e.g. Xilinx FSL), and general-purpose system busses (e.g. On-chip Peripheral Bus (OPB)). Processor local busses and direct-connection buses are processor specific and always with lower latency, while system busses are generic and slow. Since the Xilinx MicroBlaze processor has been updated to use a PLBv46 interface instead of OPB for peripheral and non-cached memory access most recently, in this research we will only discuss designs using PLB and FSL.

The PLB interface is a memory-mapped interface for peripheral components. The PLB bus is a shared, variable latency bus which incorporates a centralized bus/arbiter with multiple masters and multiple slaves attached to it. It is also used to interconnect soft- and hard-core processors in a Xilinx FPGA. The hardware side of a PLB interface consists of a decoder for a memory-read or memory-write cycle on a selected address in the memory range mapped to the PLB. The decoded memory cycle is translated to a read-from or a write-into

a register in the coprocessor. A memory-mapped interface is an easy and popular interface technique, in particular because it works with standard C on any core that has a system bus. The drawback of this interface is the low-speed connection between hardware and software. Even on an embedded core, a simple round-trip communication between software and hardware may run into several tens of CPU clock cycles [125]. Therefore, modern SoC designs are gradually moving away from this type of bus topology and instead embracing point to point interconnection schemes for high performance designs.

The FSL Bus is a uni-directional point-to-point communication channel bus used to perform fast communication between any two design elements on the FPGA when implementing an interface to the FSL bus. The FSL interface is specific to the Xilinx 32-bit software core processor, MicroBlaze. The interfaces are used to transfer data to and from the register file on the processor to hardware running on the FPGA. The FSL outperforms PLB mainly because of its three advanced features: (1) the FSL is a dedicated, non-shared link driven by a simple handshake protocol; (2) the MicroBlaze provides dedicated instructions to access this interface; (3) the FSL can be buffered with a dedicated queue, which enables execution overlap of the MicroBlaze and coprocessor operations.

3.1.3 Cosimulation-based on StrongARM

Under the GEZEL simulation environment we first implement cryptographic primitives in GEZEL-based on FSMD model. A standalone simulation is then used to verify the correctness. Next, the primitives are integrated into a coprocessor shell as follows.

PLB-based design. Three memory-mapped registers have been added: a data-input port, a data-output port, and control port. The control shell also contains a dedicated controller that controls the operations of the hardware/software interface, which is PLB interface in our design. This controller implements the 'instruction-set' for the coprocessor, and decodes the commands sent from the software driver to the coprocessor. The final step is to write a software driver to perform a series of memory reads and writes.

FSL-based design. Although the FSL can only be used on MicroBlaze, we can still model its protocol and make the emulated FSL run on the ARM ISS since a cycle-accurate MicroBlaze ISS is currently unavailable in GEZEL. Since the ARM does not have FSL we have to emulate it through a memory-mapped protocol. However, the hardware side of FSL described in GEZEL still implements the exact FSL protocol with the same FSL interface signals instantiated as a GEZEL 'ipblock'. This means any coprocessor that can be functionally verified using cosimulation with this setup, will also work when attached to the actual MicroBlaze FSL.

Both of the emulated PLB and FSL protocols are based on the memory-mapped interface. For the hardware side, instead of using an instruction decoder to interpret the commands sent through memory mapped interface in the PLB base system, the FSL coprocessor implements handshake logic to interact with ARM ISS. Fig. 3.2 and 3.3 illustrates a comparison between the cosimulation setup and the physical FPGA-based implementation for the case of FSL and PLB protocols respectively.

3.2 SoC Integration of PRESENT and AES Block Ciphers

3.2.1 Introduction

In recent years, FPGAs have had major impact on HW/SW codesign. Compared to the early frequent use as devices for rapid prototyping, FPGAs are now used for final products, thanks to their reduced time-to-market and the cost advantages of standard devices. Due to the importance of reconfigurable devices, numerous FPGA AES implementations have been published, most of which focus on high throughput rates [37, 106]. In [55], an AES design achieves a throughput of 25 Gb/s on a Xilinx Spartan-3 FPGA. This number only reflects the raw processing ability of the hardware to encrypt bits. However, FPGAs are now becoming

a popular platform for SoC designs. By providing hard and soft embedded processors on FPGAs, they enable on-chip integration of coprocessors and processors. If we re-examine the above high throughput designs in the context of a SoC system, the communication bandwidth between system components becomes a critical design factor. For example, if we only consider an AES coprocessor that runs at 100 MHz and requires 11 clock cycles per encryption round, each round requires a 128-bit key, 128-bit plaintext and 128-bit crypto text, and we need an input/output bandwidth of about 3.5Gb/s. Dedicated communication hardware (e.g. direct-memory-access (DMA) chips on fixed-latency buses) may achieve this bandwidth. In many cases however, this bandwidth needs to be provided directly through the software. The bandwidth of 3.5Gb/s indeed is outside the capability of most embedded processors [125].

This shows the most optimal hardware design (in terms of performance) may not always be the most optimal solution at system level. Therefore, most previous research results on comparison between different block ciphers as standalone hardware [121, 140] cannot be directly applied for system integrations. In fact, not only the performance, but also the power or energy efficiency should be re-considered at system-level. Some researchers have noticed this problem and published some nice results on the system-level evaluation of some cryptosystems [101, 118]. However, most of them only focus on the evaluation of crypto software and target the wireless sensor node. We believe with the growing need for information security and high performance configurable computing the system integration of various crypto coprocessors will become common in many applications. However, the selection of ciphers is not a trivial work since we have to deal with tradeoffs among area, speed, power efficiency, energy efficiency and security, and all of them have to be evaluated at system level which is not enough emphasized in the previous research. In this work, we try to provide some clues to partially solve the above problem. For detailed discussion and comparison, we consider two block ciphers, AES and PRESENT, for hardware acceleration and FPGA SoC integration, and using hardware/software interfaces provided with StrongARM and Xilinx MicroBlaze processors. The results for StrongARM are estimated using cosimulation [122, 124]. The

results for MicroBlaze have been implemented on an FPGA board and measured using a hardware timer. In addition, power estimation was performed at system level using the Xilinx XPower tool.

This part of the dissertation is organized as follows. Section 3.2.2 performs some analysis on the performance and power consumption under co-simulation environment. Section 3.2.3 describes the FPGA-based SoC design and illustrates the experimental results. Finally, Section 3.2.4 summarizes the research for the SoC integration of block ciphers.

3.2.2 Cosimulation-based Codesign Analysis

Under the GEZEL simulation environment we first implement the AES and PRESENT in GEZEL-based on FSMD model. A standalone simulation is then used to verify the correctness of the AES and PRESENT encryption core. Next, the AES and PRESENT cores are integrated into a coprocessor shell.

The simulation results for the AES and PRESENT under GEZEL environment are illustrated in Table 3.2. Note that all the designs consider the encryption only with two key management schemes: programmable key (the key changes for each AES or PRESENT encryption) and fixed key (the key is fixed and stored in coprocessor).

From Table 3.2 it is obvious that if we only consider the hardware acceleration the design speedup is often dramatic, but if we consider the system integration with either PLB or FSL bus interface and take into account the communication overhead, the resulting speedup can be much lower. A detailed timing analysis of GEZEL cosimulation can be found in Table 3.3, which can be then used to compare with the real FPGA implementation timing results in Table 3.8. Here we roughly divide the overall latency for one encryption into three major components: communication time, computation time and software overhead. For the PLB-based cosimulation, both of the AES- and PRESENT-based systems are communication constrained, which means the system is limited by the throughput of the PLB bus between

Table 3.2: Cosimulation performance results (one encryption for each block cipher)

Standalone	SW cycle count	HW cycle count	HW speedup	
AES-128	3,271	12	272.58	
PRESENT-80	19,244	33	583.15	
Programmable Key	PLB cycle count	FSL cycle count	PLB speedup	FSL speedup
AES-128	1,850	230	1.77	14.22
PRESENT-80	1,172	162	16.42	118.79
Fixed Key	PLB cycle count	FSL cycle count	PLB speedup	FSL speedup
AES-128	1,303	157	2.51	20.83
PRESENT-80	769	113	25.02	170.30

Table 3.3: Timing analysis on GEZEL cosimulation (cc: cycle counts)

Programmable Key	AES-128		PRESENT-80	
	PLB Cop	FSL Cop	PLB Cop	FSL Cop
Bus latency (cc)	44	14	44	14
Num. of instructions	29	0	19	0
Num. of data	12	12	7	7
Comm. time (cc)	1,804	168	1,144	98
Comput. time (cc)	12*	12	33*	33
SW overhead (cc)	46	50	28	31
Overall latency (cc)	1,850	230	1,172	162
Fixed Key	AES-128		PRESENT-80	
	PLB Cop	FSL Cop	PLB Cop	FSL Cop
Bus latency (cc)	44	14	44	14
Num. of instructions	21	0	13	0
Num. of data	8	8	4	4
Comm. time (cc)	1,276	112	748	56
Comput. time (cc)	12*	12	33*	33
SW overhead (cc)	27	33	21	24
Overall latency	1,303	157	769	113

*: denote the time fully overlapped with the communication time.

Table 3.4: Toggle counts (TC) of standalone simulation

	TC/cycle	TC/encryption	TC/(encryption $*$ byte)
AES-128	3,798	45,576	2,849
PRESENT-80	2,746	90,618	11,327

Table 3.5: Power results of standalone FPGA implementations (10 encryptions for each block cipher working at 20MHz)

	Quiescent Power(mW)	Dynamic Power(mW)	Time (ms)	Energy (mJ)	Energy/byte (μJ/byte)
AES-128	51.51	40.75	6	0.55	3.46
PRESENT-80	44.06	3.49	16.5	0.78	9.81

the crypto coprocessor and ARM. For example if an encryption can finish in 44 clock cycles (e.g. 12 cycles for AES-128 and 33 cycles for PRESENT-80), the computation time becomes invisible.

The GEZEL simulation environment also provides technology-independent toggle counting at RTL level, which is useful to roughly estimate the dynamic power consumption of a design.

The toggle counts collected in Table 3.4 only includes the AES and PRESENT encryption core when doing standalone simulation. This data can be utilized to estimate the power- and energy-efficiency of the hardware designs of AES and PRESENT early. The first column of the table indicates that PRESENT is more power efficient than AES, in terms of dynamic power consumption. However, since most light-weight block ciphers, like PRESENT, are specialized cryptographic implementations for tight cost constraint applications, such as wireless sensor nodes, the energy-efficiency instead of power-efficiency should be emphasized because most of these applications are battery powered. The second column of the table reflects the toggle counts per cycle multiplying the cycle counts for each encryption, the results of which can be approximately equivalent to the energy consumption per encryption.

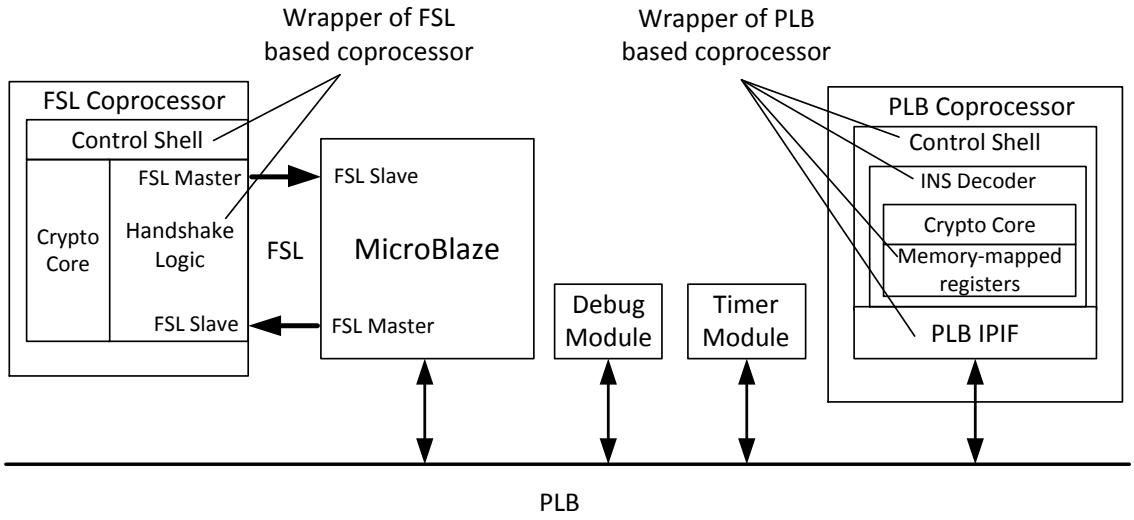


Figure 3.4: FPGA-based SoC platform.

Further, we divide the toggle counts per encryption by the number of plaintext bytes in one encryption. The obtained values can be assumed to be the energy required for the block cipher to encrypt one byte plaintext. This indicates that the PRESENT might be less energy-efficient than the AES in standalone encryption mode. Table 3.5 illustrates the power values (by using post-place & route simulation model in XPower) obtained in standalone simulation on Xilinx Spartan-3 XC3S1000 FPGA, which supports our hypothesis based on toggle counts. Moreover, it reflects the relative accuracy of GEZEL toggle counting when predicting the actual power consumption of designs. Note that both the quiescent and dynamic power values are collected from *Vccint*, the FPGA core power supply voltage since we only consider the FPGA core power variations.

3.2.3 FPGA-based Codesign Implementation

Using the above GEZEL simulation environment we can translate the GEZEL description of the AES and PRESENT and control shells into synthesizable VHDL, which can be then added as coprocessors in the Xilinx Platform Studio (XPS) 9.2.02i.

The SoC system shown in Fig. 3.4 is built on a Xilinx Spartan-3E XC3S1600E-FGG320

Table 3.6: FPGA system performance results (one encryption for each block cipher)

Standalone	SW cycle count	HW cycle count	HW speedup	
AES-128	4,655	12	387.92	
PRESENT-80	22,959	33	695.73	
Programmable Key	PLB cycle count	FSL cycle count	PLB speedup	FSL speedup
AES-128	439	59	10.60	78.90
PRESENT-80	301	61	76.28	376.38
Fixed Key	PLB cycle count	FSL cycle count	PLB speedup	FSL speedup
AES-128	309	41	15.06	113.54
PRESENT-80	207	52	110.91	441.52

Table 3.7: FPGA implementation areas (unit: slices)

Programmable Key	AES-128		PRESENT-80	
	PLB Cop	FSL Cop	PLB Cop	FSL Cop
Coprocessor w/ wrapper	2,133	2,068	511	353
Only wrapper	258	134	234	65
Fixed Key	AES-128		PRESENT-80	
	PLB Cop	FSL Cop	PLB Cop	FSL Cop
Coprocessor w/ wrapper	2,162	1,976	469	290
Only wrapper	252	104	243	67

development board with both on- and off-chip memory. Since each on-chip memory read or write only takes 2 clock cycles compared to 22 and 23 clock cycles for off-chip memory read or write on our specific FPGA platform, we fully utilize the on-chip memory for our system design. Note that our research objective is trying to address some system integration issues or considerations for general SoC systems and in this research the selection of MicroBlaze as the microprocessor and PLB (or FSL) as system bus is for detailed discussion.

To get a good indication of the power consumed by the device using XPower, an accurate VCD file is needed. Hence, we use a complete post-place-and-route, timing-accurate model

built in XPS to generate the VCD file. Other system settings use the XPower default values.

The areas for AES and PRESENT coprocessors are presented in Table 3.7. The performance numbers in Table 3.6 are for one encryption for the AES and PRESENT. The performance improvement from using hardware/software codesign is satisfying respect to the pure C software implementations (with -O2 compiler option). However, combined with the former cosimulation results in GEZEL we see that the overhead is substantial. Take AES-128 FPGA PLB-based codesign with programmable key management for example, one encryption in hardware only should take 12 clock cycles, while we have used 439. This overhead factor (37X) is due to the communication with the processor and implementation of various command sequences with the encapsulated hardware. Note that the big differences in cycle counts and speedup values between GEZEL cosimulation and FPGA SoC implementation are due to the different processors (StrongARM *vs.* MicroBlaze) and compilers (arm-linux-gcc *vs.* mb-gcc). A detailed timing comparison of the cosimulation and FPGA implementation can be found in Table 3.3 and Table 3.8.

In the previous relative power estimation using toggle counts (shown in Fig. 3.4), we deduced that the more power-efficient PRESENT block cipher is in fact less energy efficient than AES, in terms of toggles per encryption per byte together with the standalone FPGA simulation results. When we look at this problem again in an FPGA-based SoC platform, we can find that the PLB-based system with PRESENT coprocessor consumes slightly less total power than that with the AES coprocessor and becomes much less in FSL-based system. However, in both PLB- and FSL-based systems, still we can find that the PRESENT-based system is less energy-efficient than the AES-based system.

We would like to further point out the impact of bus selections on the power and energy efficiency. As indicated in Table 3.9, the FSL-based crypto system consumes more power than PLB-based one, which is obvious for the AES system, since the majority of the dynamic power should be consumed by the crypto core and the power for data preparation, including the instruction and data bus transfers, is relatively lower. In this case, the selection of faster

Table 3.8: Timing analysis on FPGA implementation (cc: cycle counts)

Programmable Key	AES-128		PRESENT-80	
	PLB Cop	FSL Cop	PLB Cop	FSL Cop
Bus latency (cc)	9	2	9	2
Num. of instructions	29	0	19	0
Num. of data	12	12	7	7
Comm. time (cc)	369	24	234	14
Comput. time (cc)	12*	12	33**	33
SW overhead (cc)	70	23	67	14
Overall latency (cc)	439	59	301	61
Fixed Key	AES-128		PRESENT-80	
	PLB Cop	FSL Cop	PLB Cop	FSL Cop
Bus latency (cc)	9	9	2	2
Num. of instructions	21	0	13	0
Num. of data	8	8	4	4
Comm. time (cc)	261	16	153	8
Comput. time (cc)	12*	12	33*	33
SW overhead (cc)	48	13	54	11
Overall latency	309	41	207	52

*: denote the time fully overlapped with the communication time.
**: denote the time partially overlapped with the communication time.

Table 3.9: FPGA system power and energy simulation results (4 encryptions for each block cipher woking at 50MHz)

Programmable Key	Quiescent Power(mW)	Dynamic Power(mW)	Time (ms)	Energy (mJ)	Energy/byte (μJ/byte)
PLB AES-128	83.91	20.70	35.32	3.69	57.73
PLB PRESENT-80	83.93	21.30	24.20	2.55	79.58
FSL AES-128	84.50	41.86	4.92	0.62	9.71
FSL PRESENT-80	83.96	22.49	5.00	0.53	16.63
Fixed Key					
PLB AES-128	84.01	24.23	24.84	2.69	42.01
PLB PRESENT-80	83.92	20.99	16.68	1.75	54.68
FSL AES-128	84.63	46.26	3.40	0.45	6.95
FSL PRESENT-80	84.04	25.18	4.28	0.47	14.61

bus system may results in higher power consumption. However, in terms of energy efficiency, the FSL-based system only spends about 20% of the energy consumed by PLB-based system to encrypt one byte plain text in average. So, the experimental results show that the bus selection is a tradeoff between power efficiency and energy efficiency.

3.2.4 Summary

Due to the encryption speed and ease of implementation, block ciphers have been widely used in various embedded applications. Much research effort has been put on the trade-off designs on hardware implementation of block ciphers, but, we think that the hardware profile is unable to predict the performance and energy (or power) in the context of a real embedded system.

Using our design flow we can not only get some early prediction of performance and dynamic power consumption under co-simulation environment, which can help designers to refine the design at an early stage, but also get accurate performance and energy values after on-board FPGA implementation, which can help designers select the crypto coprocessor best fitted to some specific platforms. The illustrated SoC designs with AES and PRESENT coprocessors and different bus protocols (PLB and FSL) identify the HW/SW interfaces design as an important system integration issue, and address the power- and energy-efficiency evaluation issue at the system level.

3.3 SoC Integration of an Elliptic Curve Cryptography Coprocessor

3.3.1 Introduction

Public-key cryptosystems, especially elliptic curve cryptography [69,91,108] and recently extensively discussed hyper-elliptic curve cryptosystems (HECC) [92], have become very popular. They have become the preferred public-key cryptosystem for many critical embedded applications. Implementing ECC on an embedded system, including both the hardware and software components, can be a real challenge since one has to deal with ultra-long bit-width data with constrained resources and processing power. A promising approach to deal with this dilemma is the HW/SW codesign which offers the advantage of flexibility in software with performance in hardware.

The computationally intensive kernel of ECC is well suited for hardware acceleration, so HW/SW codesign is the logic choice to evaluate tradeoffs between cost and performance. In the last couple of years, many researchers have used codesign techniques to explore trade-offs between cost, performance and security in ECC system designs. Typical target platforms include low-end 8-bit platforms (e.g. AVR or 8051) [2, 16, 68, 76, 96, 97] as well as 32-bit microprocessors with bus systems (e.g. MicroBlaze with PLB bus). Orlando [116] proposed a scalable elliptic-curve processor architecture which operates over the binary field GF(2^m). Gura [69] have introduced a programmable hardware accelerator for ECC over GF(2^m), which can be attached to a 64-bit PCI bus and supports field sizes up to 255. These two papers emphasize the optimization of the coprocessor for speedup and scalability. Sakiyama [120] explored architecture optimizations for ECC coprocessors and showed how to exploit parallelism using local control and local coprocessor storage. Their experiments were based on ARM cosimulation. Although each of the above three papers discussed coprocessor implementation results for FPGA, none of them presented a detailed discussion of the system integration effects and the impact of communication bottlenecks when attached

to actual processors. Cheung [35] implemented a coprocessor with parallel field multiplier attached to OPB bus and identified data-transfers over the processor bus as an important system integration problem, but no further optimization for this was mentioned.

Although the design goals in 8-bit platforms and platforms with 32-bit microprocessors and bus systems may differ due to different applications (e.g. low power sensor nodes *vs.* high performance security systems), both of them have to deal with the same problem of how to minimize the communication overhead resulting from using a single, central controller.

Compared with the previous work, this work presents four contributions to codesigns for ECC.

First, we present a complete ECC SoC design and focus on the system integration issues on a real FPGA platform. We use HW/SW co-simulation to do system profiling of the bus bottleneck, and we explore multiple control hierarchies in a typical FPGA-based SoC system. We show that, using proper partitioning of control and data, the ECC system execution time can be made almost independent of the selection of bus protocols between the central controller and the coprocessor. Also, we present system performance profiles for an ECC SoC design with MicroBlaze processor and PLB (Processor Local Bus).

Second, we identify two system performance regions. In the first region, the overall system is performance-constrained due to the coprocessor hardware. In the second region, the overall system is communication-constrained due to the coprocessor-CPU bus. The actual operating point of the system is determined by the choice of coprocessor configurations.

Third, we quantify the impact of control hierarchy and local storage in the coprocessor, and show how the system performance regions are affected. Specifically, we use a small 8-bit microcontroller, PicoBlaze, as a local control unit inside the coprocessor. Moreover, we optimize the control hierarchy by converting a Single-Picoblaze sequencer architecture into a Dual-Picoblaze architecture which runs interleaved instruction sequences. This Dual-PicoBlaze based architecture can achieve the instruction transfer rate of 1 instruction/cycle, while a Single-Picoblaze architecture only provides half that speed, 1 instruction per 2 cycles.

The FPGA implementation results show that our proposed ECC SoC architecture with the Dual-PicoBlaze based coprocessor has a better tradeoff between area and speed.

Finally, we optimize the ECC coprocessor to have high flexibility by proposing a novel parallel architecture, which can be used to explore application- and algorithm-level parallelism of ECC. The architecture is scalable and can be adapted to different bus interfaces. Based on our theoretical analysis on its scalability, the parallel ECC SoC design can be used for various high performance applications.

The remainder of this section is as follows. Section 3.3.2 gives a brief description of our ECC system configuration and the definition of the ECC design space in our design. In Section 3.3.3, the problem statement of HW/SW partitioning will be given. The implementation details will be discussed from Section 3.3.4 to Section 3.3.7. Section 3.3.8 explains the HW/SW codesign flow used in this work, and performance results of FPGA implementations are analyzed. Section 3.3.9 concludes the research for SoC integration of ECC coprocessor.

3.3.2 Basic Configurations and Design Space

A basic building block of all elliptic curve cryptosystems is the scalar multiplication, an operation of the form $k \cdot P$ where k is an integer and P is a point on an elliptic curve. A scalar multiplication can be realized through a sequence of point additions and doublings. This operation dominates the execution time of cryptographic schemes based on ECC, such as signatures (ECDSA). There are many design options for ECC implementations, including the coordinate system, the field and the type of curve used [72]. In this work, we started from a basic configuration as shown in Table 3.10.

We consider design space exploration for ECC system architecture at two abstraction levels.

At the application-level of ECSM, we discuss the scalability of our proposed parallel ECC coprocessor architecture. Multiple scalar multiplications can be computed simultaneously on

Table 3.10: Basic Configuration for the proposed ECC coprocessor Design

Coordinate System	L-D Projective coordinates [104]
Point Mult.	Montgomery Scalar Mult. [104]
Field	GF(2^{163})
Curve	Koblitz curve (B-163)
GF Mult.	Bit- /Digit-serial multipliers [57, 119]
GF Addition	Logic XOR operations
GF Square	Dedicated hardware with square and reduction circuits [72]
GF Inversion	GF Multiplications and Squares based on Fermat's Theorem [117]

one parallel ECC coprocessor, and each of them is executed on independent ECC datapath.

At the algorithm-level of ECSM, we discuss the mapping of existing parallel ECSM algorithms [82] to our parallel ECC architecture, and one scalar multiplication result can be obtained by summing all the partial ECSM results from parallel datapaths.

At the low abstraction level of finite field arithmetic, the design space is defined by the use of different field multiplier architectures, including bit-serial as well as digit-serial multipliers of different sizes. A basic bit-serial multiplication in GF(2^m) can be realized through a classic 'shift-and-XOR' based MSB-first bit-serial multiplier with interleaved reduction modulo the irreducible polynomial [57]. It can finish one GF(2^{163}) multiplication in 163 clock cycles. A digit-serial multiplier on the other hand can process multiple bits of the operands in parallel with a processing time proportional to $\lceil m/D \rceil$ cycles, with digit size $D \leq m - k$, where m is 163 and k is 7 for the B-163 curve. It is obvious that within a certain range of D, when increasing the D, the area will increase accordingly, but the processing time will be the same. For example, for all $D \in [55,81]$, the multiplication time is 3 clock cycles. In this case we only select D size of 55 for our implementations.

Evaluating the multiplication speed and the area-time product for the different architec-

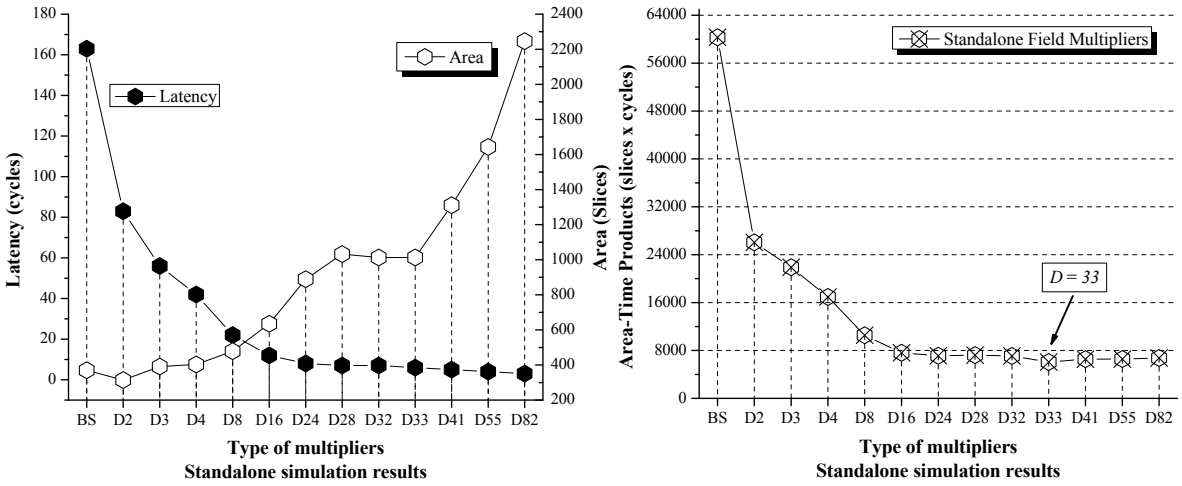


Figure 3.5: Area, time and area-time products of different multiplier implementations.

tures leads to the optimum digit size for an implementation on a specific platform. We have evaluated the multipliers required for the NIST B-163 polynomial for different digit sizes to find the optimum values. This leads to Fig. 3.5, which shows the post-place and route results for a Virtex-5 XC5VLX50-1FF676 FPGA with Xilinx ISE10.1 with default settings. For the area metric in Virtex-5 FPGA, we use the unit, *slice*, which is based on the new six-input LUTs. The FPGA implementation results may be affected by different system constraints after place and route and different FPGA platforms.

Even though a digit-serial multiplier was presented in [98], the conclusions of those authors cannot be directly applied here. The differences are due to the use of different technologies (ASICs *vs*. FPGA) and target application (standalone components *vs*. system coprocessors). In our case, the system clock frequency is 125MHz, determined by the MicroBlaze system processor. Even though we find that the multipliers can run above 200MHz, we will operate them at the 125MHz system clock. Indeed the coprocessor may run at higher speed, but that would complicate the system implementation by introducing multiple clock regions. Therefore, instead of using absolute execution time and equivalent gate counts, we use cycle counts and total used slices to calculate the area-time products. Then, from Fig. 3.5 we can identify that the best choice, in terms of the area-time product, for a field multiplier

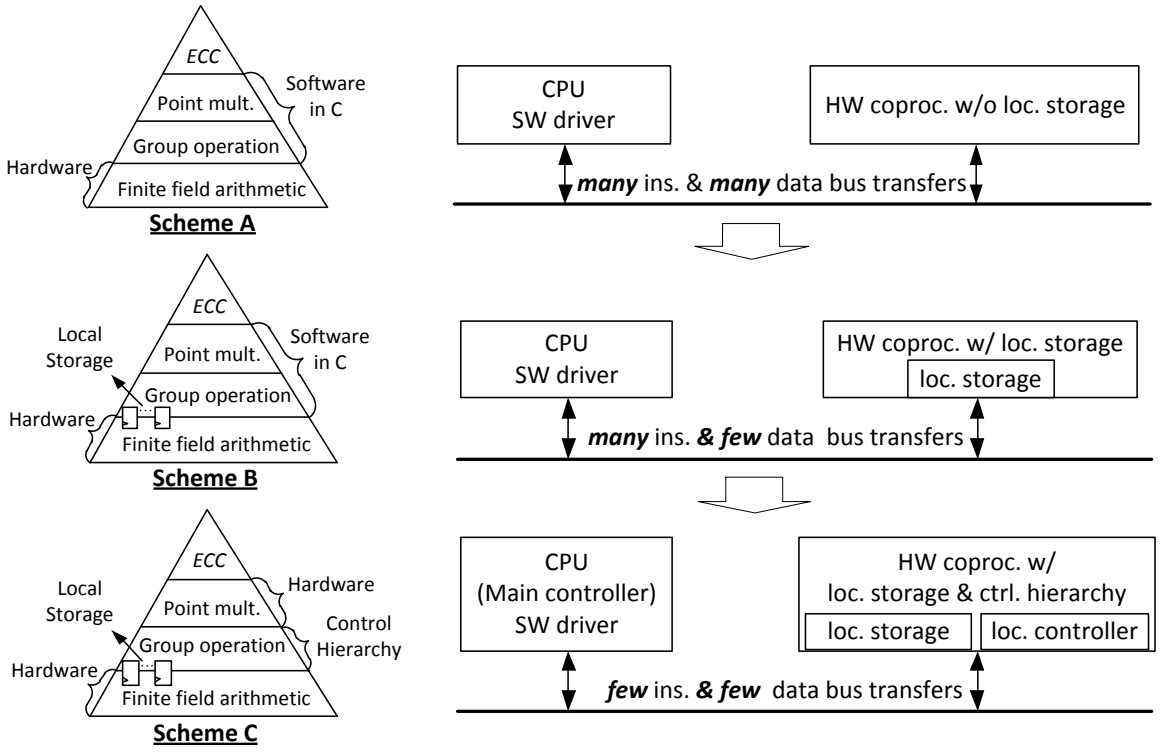


Figure 3.6: System architecture modeling of different schemes.

in a stand-alone design is a digit-serial multiplier with D size of 33. All the bit-serial and digit-serial multipliers are implemented in logic without using hardware macros because field multiplications on binary fields are mainly composed of shift and XOR operations. Also, the multipliers cannot be pipelined due to the iteration structure and data dependency inside of the bit- and digit-serial multiplier algorithms.

3.3.3 Considerations for ECC HW/SW Partitioning

As shown in Fig. 3.6, a scalar multiplication, $k \cdot P$, with an integer k and a point P on an elliptic curve, needs to be realized through a sequence of point additions and doublings. These group operations can be further decomposed into several types of finite field arithmetic with ultra-long operand word length (e.g. 163-bit or above).

Several researchers [96, 120] have proposed to implement the field multiplication in hard-

Table 3.11: System profiling from GEZEL cosimulation

# Access 163-bit local reg.	Baseline Design bus transactions		Uni-PicoBlaze bus transactions		Dual-PicoBlaze bus transactions	
	# Ins.	# Data	# Ins.	# Data	# Ins.	# Data
2,788	26,791	1,294	481	489	468	476

ware and the upper-level point multiplication in software. This typical partitioning is a tradeoff between flexibility, cost and speed, which are the cost factors of most importance in embedded ECC implementations. However, this partitioning may result in a HW/SW communication bottleneck since the lower-level field multiplication function will always be called by upper-level point operations, including many instruction and data transfers. For a baseline ECC system in our design (Scheme A in Fig. 3.6), with all parameters and intermediate results stored in processor local memory, the data/instruction transfer time may take up to 98.3% of the total time required to perform one GF(2^{163}) scalar multiplication when using digit-serial multiplier with D (short for digit) size of 82 and a typical bus communication latency of 9 clock cycles [59]. So, the overall ECC system speedup brought by increasing the D size of digit-serial multipliers would be buried if we cannot optimize the hardware/software communication bottleneck.

From the above analysis on the HW/SW partitioning, we already know where the system bottleneck will be, so before starting the system-level design we should first quantitatively measure the bus transactions in the baseline design, and estimate the optimization room left for us. Table 3.11 shows the system profiling from GEZEL cosimulation. Since our system bus interface uses the memory-mapped registers, the way we collect the number of instruction and data transfers over the bus is to measure how many write and read on these registers for instruction or data transfers.

3.3.4 Proposed Optimizations

Targeting the above communication bottleneck problem, we tried to optimize the HW/SW boundary in two steps: reducing data transfers and accelerating instruction transfers.

Apart from architecture-level optimizations (Scheme B and C in Fig. 3.6), we also evaluated the impact of algorithm-level optimizations to further improve the original Montgomery Scalar Multiplication algorithm [104]. These algorithmic optimizations focus on the data and instruction transfers. First, by adding I/O local registers into the coprocessor, data can be transferred between the software and the coprocessor while an ECC field multiplication is in progress. Second, the local registers used to store operands also enable data reuse in the coprocessor. For example, a field multiplication is always followed by a field addition, so the field multiplication results can be directly used as one operand for the following addition. By exploiting the data dependencies in the point multiplication and coordinate conversion operations, we can avoid additional data transfers between the CPU and the coprocessor.

3.3.5 Impact of Local Storage

Scheme A: using Processor Local Memory as main storage. In a straightforward design, like Scheme A in Fig. 3.6, the ECC system will implement the point operations on the main processor. This requires storing all parameters and intermediate results in Processor Local Memory, so that the main processor can access and manipulate them. This scheme is also called the Baseline ECC SoC System.

From Table 3.11, it is observed that for a full 163-bit scalar multiplication, there are 2,788 times read/write on eight 163-bit coprocessor local registers, so if all parameters and intermediate results are stored in main memory, this may result in 16,728 times data transfers over the 32-bit bus. This represents a significant amount of time (around 60% of the total execution time of a point multiplication, assuming the typical PLB bus HW-SW latency of 9 clock cycles). Hence, a simple optimization can be achieved by adding local storage

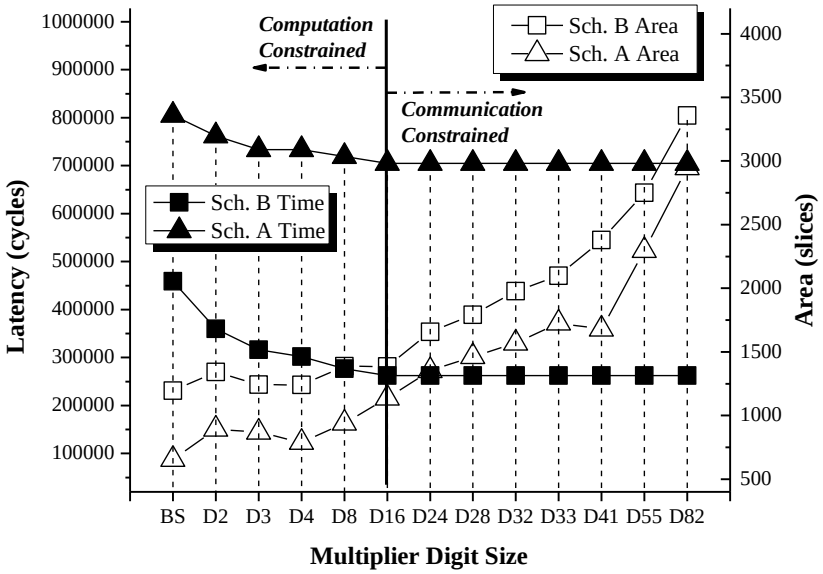


Figure 3.7: Time to complete one multiplication and area for each type of coprocessors.

to the coprocessor, like Scheme B in Fig. 3, so that the amount of data transfers over the processor-to-coprocessor bus can be minimized.

Scheme B: using Coprocessor Local Registers as local storage. We optimize the ECC baseline design by adding local storage to the coprocessor, so that the amount of data transfers over the processor-to-coprocessor bus may be reduced. In total, five 163-bit registers were added to the coprocessor. It is also possible to use SRAM blocks instead of registers to save slices.

The comparison of FPGA implementation results between Scheme A and B can be found in Fig. 3.7. Comparing Scheme A with B, Scheme B provides an average speed up of 2.5 times at the expense of a 1.3 times larger area. The figure also shows that beyond digit-size D = 16, the latency of the overall point multiplication does no longer decrease. Thus, the digit-size D = 16 splits the design space into two. The left half of the figure is a computation constrained area. In that part, the system is constrained by the efficiency of the coprocessor hardware. The right half of the figure is communication constrained. In that part, the system is limited by the processor. For example, if a multiplication in hardware can finish

in 9 clock cycles (e.g. 8 clock cycles for digit-serial multiplier of D size of 24), the speedup of standalone field multiplication brought by D sizes beyond 24 become invisible. From this point of view, we can conclude that the best hardware design may not result into the best system solution when system integration overhead is considered.

3.3.6 Impact of Control Hierarchy

Scheme C: using PicoBlaze (PB) as control hierarchy. From the above analysis of Scheme A and B, these two schemes mitigate the overall bus communication overhead by optimizing the data transfer side; however, instruction transfers still dominate the entire scalar multiplication time. From the cosimulation profiling, we can see that for a full 163-bit scalar multiplication, there are 26,791 times instruction transfers though the data transfers have been reduced to 1,294 (mostly composed of reading status registers) with a typical PLB bus communication latency of 9 clock cycles. One area for further optimization is that of coprocessor control. Indeed, for each operation performed by the coprocessor, the processor needs to perform a command transfer over the PLB bus. These command transfers are still needed, even after local registers are added to the coprocessor. In order to reduce the amount of command transfers, we must change the way to control the coprocessor.

The PicoBlaze microcontroller is a compact, capable and cost-effective fully embedded 8-bit RISC microcontroller core optimized for Xilinx FPGAs. It has predictable performance, always two clock cycles per instruction, and 1K instructions of programmable on-chip program store, automatically loaded during FPGA configuration. It only costs 53 slices and 1 block RAM on Virtex-5 XC5VLX50 FPGA and can run at the max frequency of 180MHz.

By introducing a local control hierarchy, the 8-bit microcontroller PicoBlaze takes the charge of sending out the point addition and doubling instructions. The main 32-bit microcontroller, MicroBlaze, only needs to start a scalar multiplication once, after which the detailed sequencing will be completed by the PicoBlaze (like Scheme C in Fig. 3.6). As shown in Fig. 3.8, the coprocessor has two separate FSMs to decode the instructions sent from the

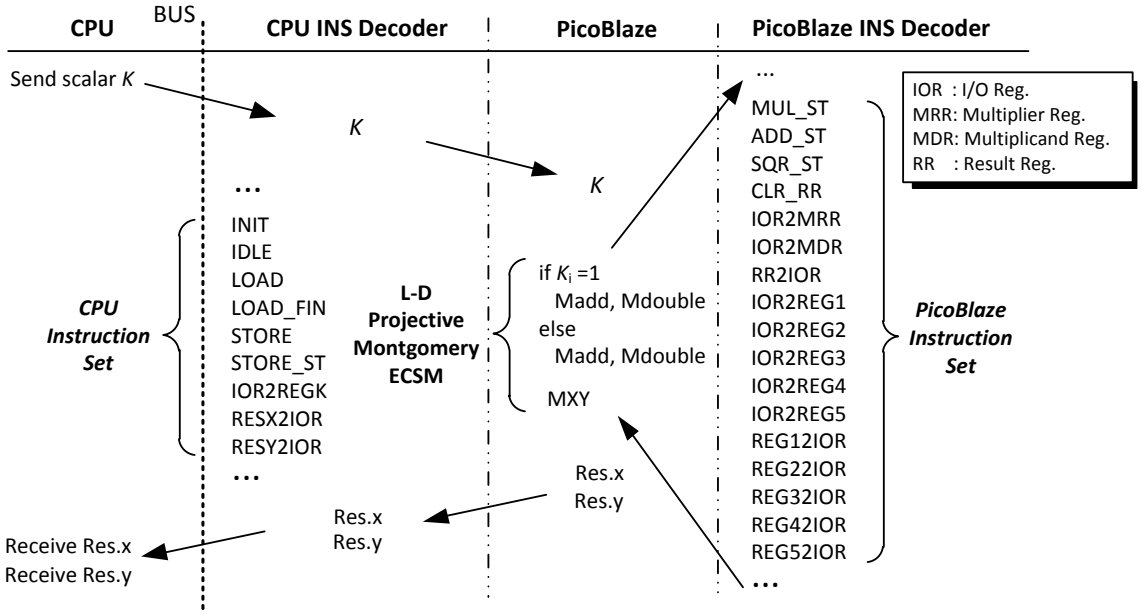


Figure 3.8: Dataflow of ECSM with CPU and PicoBlaze instruction sets.

CPU and PicoBlaze, respectively. The CPU instruction set only contains instructions for data transfers, while the PicoBlaze instruction set includes all the instructions for controlling the finite field arithmetic ALUs and exchanging data with coprocessor local register array.

The PicoBlaze has additional advantage of having a fixed instruction rate (2 clock cycles per operation). This means that the local instruction decoder in the coprocessor can be simplified: no additional synchronization is needed between the PicoBlaze and the local instruction decoder FSM. From the co-simulation system profiling , we can see that for the Single-PicoBlaze design the instruction and data transfers have been greatly reduced to 481 and 489, respectively. The lower-bound on the amount of instruction and data transfers is around 18 and 6, respectively. Most of the current measured instruction and data transfers are devoted to polling the status registers with associated instructions during the coprocessor execution time. Therefore, a simple further optimization can be conducted by using 'interrupt' control, and this improvement may not only reduce the number of bus transactions but also save the task load of ECC on MicroBlaze and PLB bus, which then may be used for other peripherals in a large system.

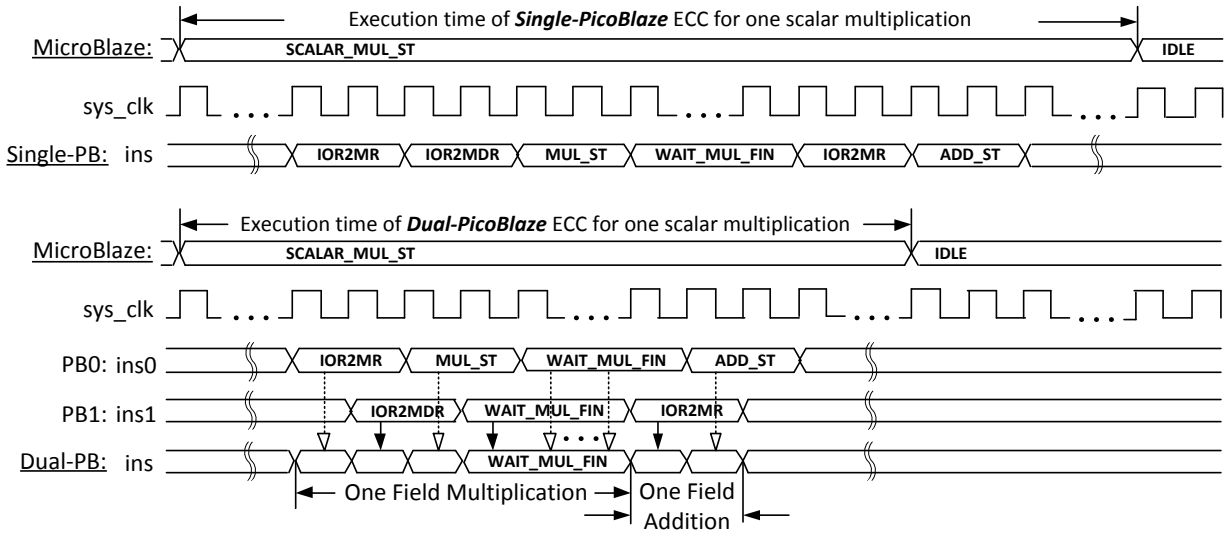


Figure 3.9: An example of interleaving PicoBlaze instructions.

The results for Scheme C with a Single-PicoBlaze local controller will be discussed and compared with other design strategies. We have demonstrated that by introducing the Picoblaze as control hierarchy the communication constrained area disappears as we can observe a continuous speedup when the coprocessor uses a faster multiplier, from a bit-serial multiplier to a digit-serial multiplier of D-size 82.

After analyzing the above design we find two characteristics of Single-PicoBlaze based ECC coprocessor design, which can help us to further refine the design. First, the local control unit, PicoBlaze, in current design is acting as an instruction generator. It has no data processing inside, which means it is feasible to split the PicoBlaze program into several subsets as long as the sequence of the instructions is guaranteed. Second, the instruction rate of Single-PicoBlaze is fixed 2 clock cycles per operation, which means that there is still one cycle per operation wasted in the ECC datapath. So, from the above two observations we propose the idea of optimizing the local control unit by converting a Single-PicoBlaze sequencer architecture into a Dual-PicoBlaze architecture which runs interleaved instruction sequences. Hence, this novel Dual-PicoBlaze based architecture can achieve the maximum instruction transfer rate of 1 instruction/cycle. To illustrate the conversion from Single-

PicoBlaze to the Dual-PicoBlaze, a simple example of PicoBlaze assembly codes executing one field multiplication followed by an addition is shown in Fig. 3.9. Compared with the Single-PicoBlaze design our proposed Dual-PicoBlaze design can save additional 18% of total clock cycles in average, at the expense of very small hardware overhead (81 slices in average from FPGA implementation results on XC5VLX50).

Traditional ways can also achieve the maximum instruction rate of 1 instruction/cycle, such as implementing the point operations in FSM or using a micro-coded controller with preset micro-codes [120].

Compared with the first approach using FSMs, our Dual-PicoBlaze architecture is more flexible and efficient. In general, the field operations can be very fast (a digit-serial multiplier with D size of 82 can finish one 163-bit field multiplication in 2 clock cycles) and a big performance gain of the whole underlying ECC system can only be obtained if new point operation algorithms with faster point operations are proposed. In this case, by fixing the lowest level field operations in hardware, updating an ECC system is just replacing the software assembly codes in PicoBlaze with the new point operation algorithms without the need to rewrite the HDLs. In [72], more than 10 point multiplication algorithms are compared and the research in this direction is still very active. In addition, this method can also enable the integration of the latest countermeasures against side-channel attacks into the algorithm for scalar multiplication.

Compared with the second approach using micro-coded controller, the Dual-PicoBlaze architecture is much easier to be programmed. The micro-coded controller needs sometimes complex dedicated controller with FSMs to dispatch instructions. Based on the Dual-PicoBlaze architecture we can simply use several PicoBlaze instructions to achieve efficient communication and synchronization with the hardware decoder without additional logic.

Using the GEZEL co-simulation environment we can translate the GEZEL description of the ECC datapath and control wrappers into synthesizable VHDL, which can be then added as coprocessors in the Xilinx Platform Studio (XPS) 10.1. The SoC system shown in

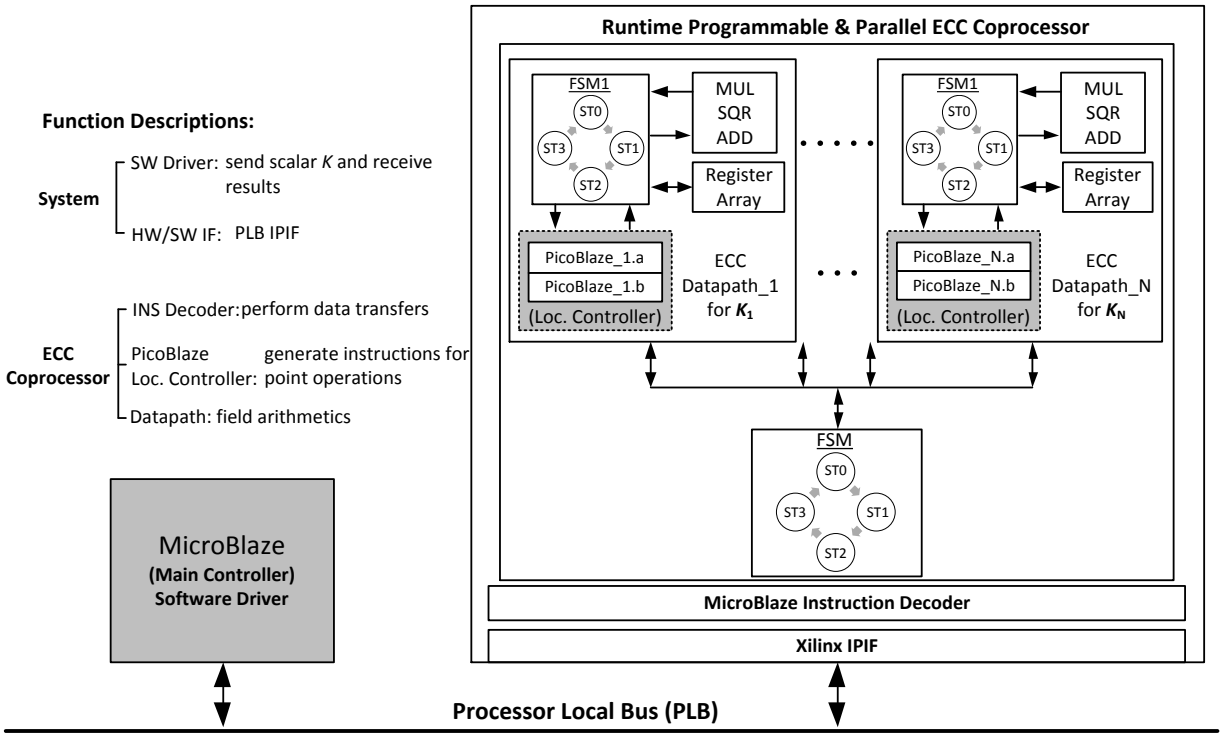


Figure 3.10: The structure of our proposed parallel ECC coprocessor.

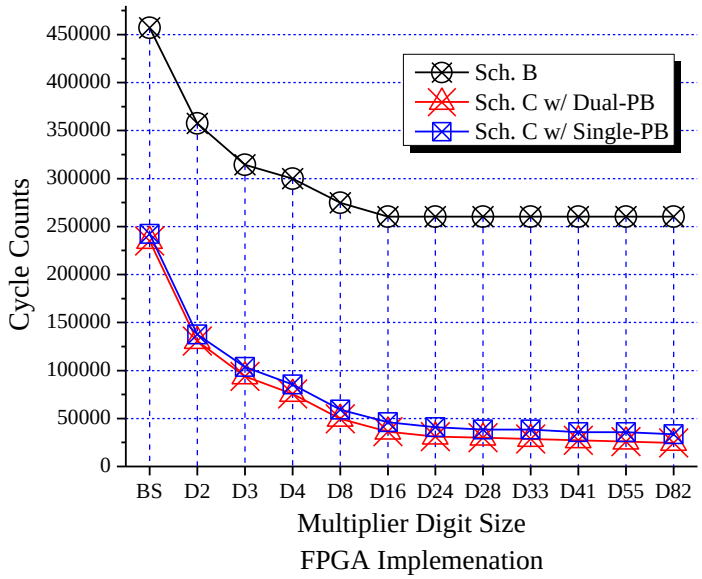


Figure 3.11: Cycle counts of FPGA implementations of each configuration of coprocessors for one full scalar multiplication.

Fig. 3.10 is built on Xilinx Virtex-5 XC5VLX50 ML501 development board.

As indicated in Fig. 3.11, for the Scheme B with only local storage, the two systems are limited by the throughput of the PLB bus. For example, for the FPGA implementation with the PLB bus latency of 9 clock cycles, if a field multiplication in hardware can finish in 9 clock cycles (e.g. 8 clock cycles for digit-serial multiplier of D size of 24), the speedup of standalone field multiplication brought by D sizes beyond 24 become invisible. For the results for Scheme C with a Single-PicoBlaze local controller, we can observe a continuous speedup when the coprocessor uses a faster multiplier, from a bit-serial multiplier to a digit-serial multiplier of D size 82. Compared to the results of Scheme B, the communication bottleneck has disappeared as we can observe a speedup of 1.2 for the coprocessor with D size 82 over the one with D size of 16. Since the control hierarchy optimization by introducing PicoBlaze also features small hardware overhead, from Fig. 3.12 we can see our proposed Dual-PicoBlaze based design can achieve the best trade-off design with D size of 28.

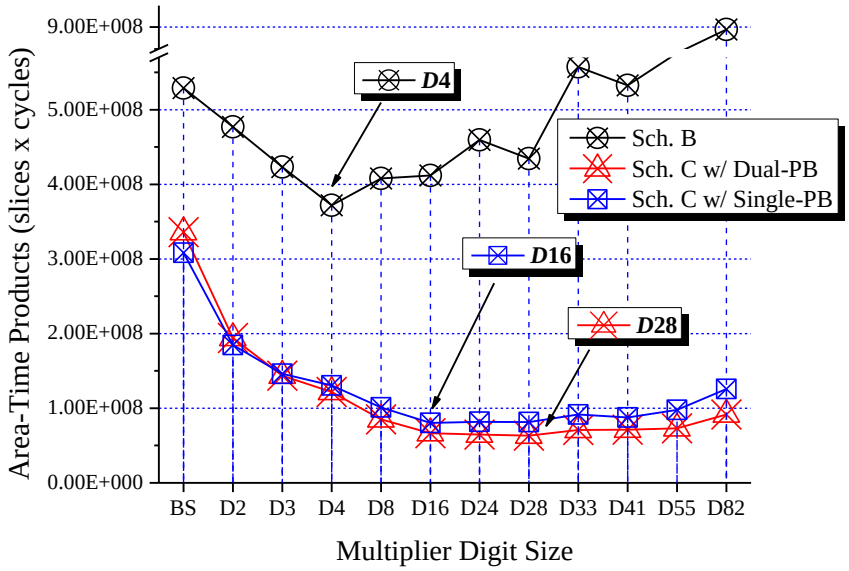


Figure 3.12: Comparison of time-area products for each configuration of coprocessors.

3.3.7 Programmability and Scalability

For our proposed structure, the CPU (main control) is not only able to send data/instructions through the bus, like the controller in most of the hardware/software codesigns, but also to program the instruction sequencer as a sub-controller in the coprocessor. The coprocessor consists of a CPU instruction decoder and single/multiple ECC datapaths, and each ECC datapath is composed of an instruction sequencer, a dedicated instruction decoder, ALU and local memory. Each ECC datapath can be programmed to carry out field operations independently.

However, the design of an instruction sequencer in the ECC datapath can be tricky. Since we have defined it to support the programmable feature, the direct use of hardware FSMs does not work. Another option is using a microcoded controller. However, the design of a dedicated controller with FSMs to dispatch instructions from microcoded controller itself can still be complex and inflexible. Finally, we come to a solution by customizing an existing low-end microcontroller to meet our requirements.

This programmable architecture gives us the freedom to efficiently utilize various counter-

measures against different side-channel attacks, which will be further discussed in Section 5. For example, we can program the sub-controller component so that it performs Montgomery ladder in order to thwart Simple Power Analysis (SPA) attacks. We can easily add base point randomization to it in order to thwart Differential Power Analysis (DPA) attacks. Finally, if the implementation requires resistance to fault attacks, we can update the program in the sub-controller to add coherence check [41] and so on. In short, the flexibility of programmable sub-controller makes the coprocessor able to update with the latest countermeasures.

For scalability, we mainly discuss about the parallelism inside of our proposed ECC coprocessor architecture.

Given the flexibility of the PicoBlaze local control hierarchy, switching the computation mode from Application-Level Parallel ECSM to Algorithm-Level Parallel ECSM, we can just modify the PicoBlaze assembly codes in each datapath without any hardware change.

Application-Level Parallel ECSM. Our proposed ECC coprocessor architecture is scalable for parallel implementations because of three design considerations: 1. distributed data processing makes each ECC datapath be independent to each other; 2. local storage makes all the initialization parameters and intermediate results be stored locally without data transfers through bus; 3. additional hierarchy of control makes point operation instructions be sent from the local control, instruction sequencer. In summary, the ECC datapath can execute scalar multiplication almost independent of the bus selections and CPU, and once the CPU send the scalar k to each ECC datapath to initialize the computation, the datapath will work automatically and turn out the right results. The maximum number of independent ECC datapath which can be attached to the CPU instruction decoder is dependent on the bus latency. Therefore, CPU can control one ECC coprocessor with N datapaths, and N point multiplications can be performed at the same time.

According to the iteration structure shown in Fig. 3.13, we can derive an equation to express the relation between the maximum number of parallel ECC datapaths and bus latency. The basic idea is to overlap the communication time with the computation time.

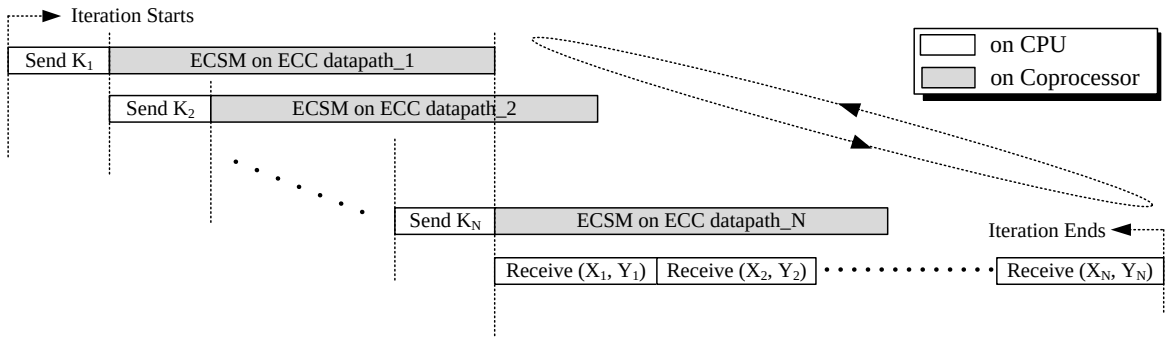


Figure 3.13: Exploration of the application-level parallelism within the proposed generic coprocessor architecture.

We assume the bus latency is T_{delay} cycles per transfer, and scalar k and results $(X,\ Y)$ each needs the same M times bus transfers (including both instruction and data transfers), and the ECSM on one ECC datapath requires T_{comp} cycles to complete, so the effective maximum number, N_{max}, of parallel ECC datapath can be expressed as

$$N_{max} = (T_{comp}/MT_{delay}) + 1. \tag{3.1}$$

From Fig. 3.13, we can observe that the results from the first datapath are ready to be sent back just after the datapath_N_{max} receives the k. Due to this parallel architecture, we can get the fastest implementation with T_{avg_min} cycles, where

$$T_{avg\_min} = 3MT_{delay}. \tag{3.2}$$

From the above equation, we can observe that for the fastest ECC coprocessor configuration with maximum number of parallel ECC datapath, the minimum computation time in average is only related to the bus latency. Also, we can have tradeoff designs between area and speed with different number of parallel ECC datapath to fit for different embedded applications, and then we can get the computation time in average, T_{avg} as

$$T_{avg} = \frac{(2N+1)MT_{delay} + T_{comp}}{N}. \tag{3.3}$$

Algorithm-Level Parallel ECSM. If we assume that a fixed polynomial is used with fixed base point, the algorithm-level parallelism inside of one scalar multiplication can also be explored. We can apply fixed window scalar splitting algorithm [82] based on our parallel ECC architecture with multiple datapaths.

If the scalar k has m bits, it can be split into N blocks using predefined windows with a size of w. Here N can be the number of parallel datapath and w is equal to $\lceil \mathrm{m/N} \rceil$. Then, the scalar k is decomposed into several parts with $k1$ consists of the w least significant bits (LSB) of k, $k2$ contains the next w bits, etc. The base point Pj for each segment of k can be pre-computed because the window sizes are fixed,

$$Pj = 2^{(j-1)w}P. \tag{3.4}$$

The overhead of pre-computing Pj can be neglected because the base point P is assumed to be fixed in most ECC applications. After the computation of each parallel ECC datapath for $kj \cdot Pj$, point additions will be followed to obtain the final results of $k \cdot P$. As a case study of two datapaths shown in Fig. 3.14, the fixed window scalar splitting algorithm is used to split the scalar k into two parts, $k1$ and $k2$, and compute them on two datapaths in parallel with pre-computed base points, $P1$ and $P2$. Only one final point addition is performed to calculate the sum of the two intermediate scalar multiplication results, $k1 \cdot P1$ and $k2 \cdot P2$. We can also clearly see that when switching the mode from Application-Level Parallel ECSM, we may just add PicoBlaze instructions for one point addition in Affine Coordinate on the first datapath and two instructions on the second datapath for moving the partial ECSM results to the first datapath.

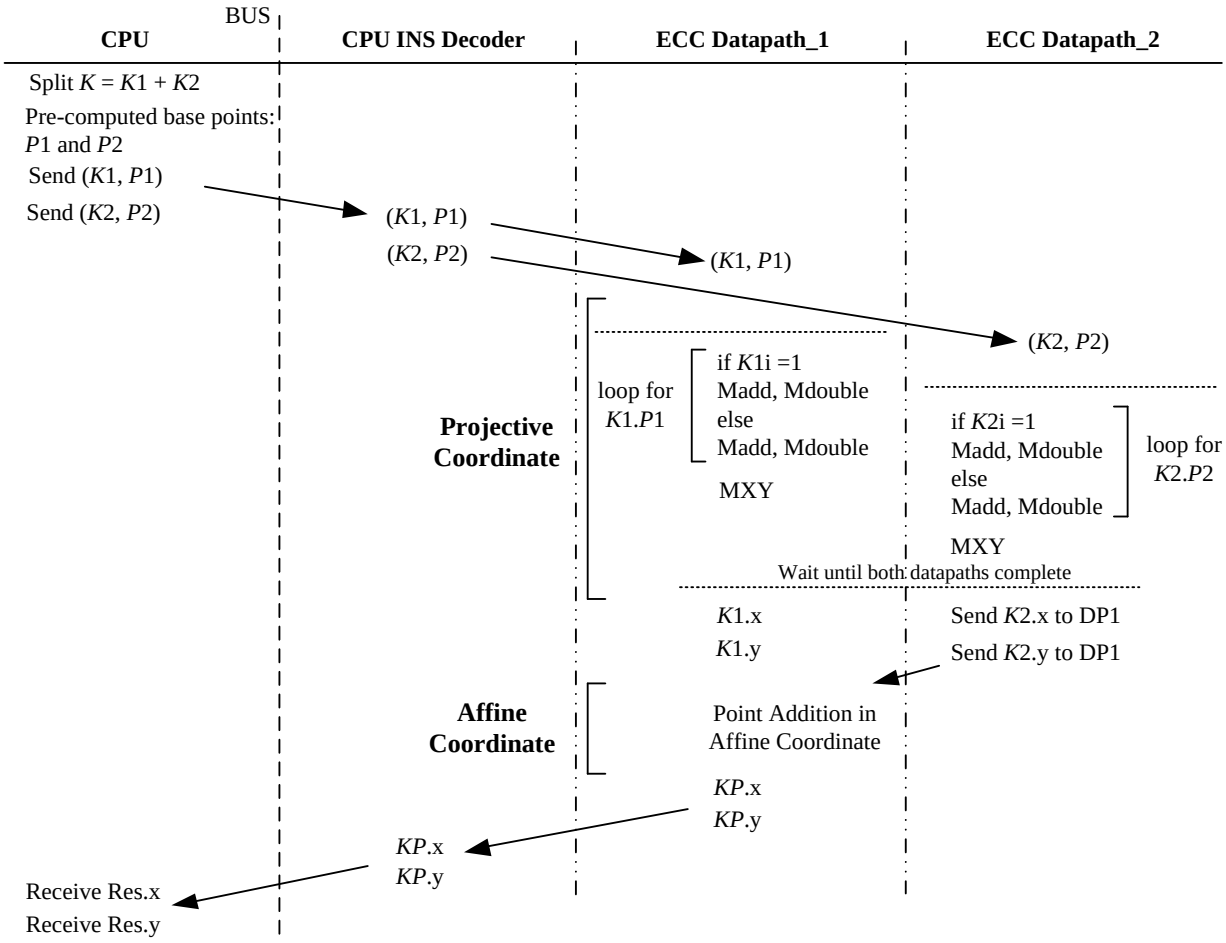


Figure 3.14: Parallel scalar multiplication with fixed window scalar splitting method.

Table 3.12: Comparison of ECC coprocessor implementations on Xilinx XC2VP30 FPGA

	Field	Platform	Slices	Cycle Counts	Field Size	Comments
Dual-PB w/ D28	GF(2^{163})	V2Pro	5,158	29,897	Fixed	Best trade-off
Dual-PB w/ D82	GF(2^{163})	V2Pro	8,944	24,689	Fixed	Fastest
ref.1 in [120]	GF(2^{163})	V2Pro	4,749	48,800	Arbitrary	1xMALU163
ref.2 in [120]	GF(2^{163})	V2Pro	8,450	28,000	Arbitrary	2xMALU163

3.3.8 Discussion of Experimental Results

In order to make a fair comparison with other published results, we also synthesize our ECC coprocessor design with single datapath based on Virtex-2 Pro XC2VP30 FPGA. As shown in Table 3.12, our Dual-PicoBlaze based ECC (with maximum frequency around 136MHz on XC2VP30-FF896-7C) shows a better trade-off between cost and performance: compare our fastest design with ref. 2, it gains 13.4% speedup; compare our best trade-off design with ref.1 and ref.2, its area-time product is 60.4% and 63.7% smaller. Although the current version of Dual-PicoBlaze design does not support arbitrary field size and superscalar as [120], it still offers an ideal alternative since in most cases the arbitrary field size is not required. This is especially true for reconfigurable computing since the hardware complexity resulted from supporting arbitrary field size in traditional ASICs can be replaced with multiple configuration bitstreams and dynamic reconfigurations. The optimizations of ECC SoC design can be done in several levels (e.g. architecture, algorithm, and circuit) and the results shown here might not be the optimal ones even in terms of the area-time product since the performance optimization focus in this work only lies on the architectural-level.

In order to show the scalability of our architecture, we also compare the design with Dual-PicoBlaze Single-Datapath ECC design with the Quad-PicoBlaze Dual-Datapath ECC designs under different operation modes. We select two extreme coprocessor configurations, the smallest (with bit-serial multiplier, BSMUL) and the fastest (with D82), for detailed

Table 3.13: Parallel ECC coprocessor implementations on Xilinx XC5VLX50 FPGA

	Field	Slices	Cycle Counts	Modes
Dual-PB Single-DP w/ BS	GF(2^{163})	1,261	234,985	Normal Operation
Dual-PB Single-DP w/ D82	GF(2^{163})	3,522	24,689	Normal Operation
Quad-PB Dual-DP w/ BS	GF(2^{163})	2,179	117,972	Application-Level Parallel
Quad-PB Dual-DP w/ D82	GF(2^{163})	6,585	12,824	Application-Level Parallel
Quad-PB Dual-DP w/ BS	GF(2^{163})	2,179	204,390	Algorithm-Level Parallel
Quad-PB Dual-DP w/ D82	GF(2^{163})	6,585	19,461	Algorithm-Level Parallel

comparison. For the results shown in Table 3.13, the cycle counts for Application-Level Parallel ECSM mode are the average speed for two ECSMs in parallel. For the results of algorithm-level Parallel ECSM, the speedup over the Dual-PB Single-DP based design is not as good as in [82] because different finite field inversion algorithms are used (Fermat's little theorem *vs.* Itoh-Tsujii [80]). However, these experimental results still effectively demonstrate the capability of our proposed architecture to switch operation modes by just revising the PicoBlaze assembly codes.

3.3.9 Summary

ECC SoC designs may become performance limited due to coprocessor data- and instruction-transfer bottleneck. The introduction of local storage and control hierarchy into the ECC coprocessor datapath can greatly reduce the communication overhead faced by traditional centralized control scheme. Starting from the system profiling of ECC codesigns using cosimulation, we tried to not repeat the conventional optimization techniques on bus communication, but instead explore new system architectures with multiple control hierarchies. This results in the Single-PicoBlaze based ECC coprocessor design and further into the Dual-PicoBlaze based design with the maximum instruction rate of 1 instruction/cycle. For flexibility, the PicoBlaze controller allows us to configure its instruction RAM and update the coprocessor

with the newly developed scalar multiplication algorithms and security countermeasures. Scalable application-level parallelism and algorithm-level parallelism can also be explored to achieve tradeoff designs between area and speed. With flexibility, ease of integration of multiple PicoBlazes into current FPGA systems and predictable performance, the proposed parallel ECC coprocessor architecture can not only be extended to other curve-based cryptography systems, but also to some other similar computationally intensive embedded applications.

3.4 Conclusion

The SoC integration of AES and PRESENT block cipher presents a basic HW/SW codesign paradigm. The entire block ciphers are implemented in hardware as a coprocessor and the software driver running on the microprocessor is only in charge of distributing key and sending messages. For much more complex public-key algorithms (e.g. ECC), we show more flexibility in the HW/SW partitioning. As a result, the impact of HW/SW interfaces will play a more important role in the design efficiency since the HW/SW communication will also become an integral part of the cryptographic computation process.

Chapter 4

Hardware Evaluation of SHA-3 Candidates

In the NIST SHA-3 competition, four typical criteria are taken into account in the evaluation of candidates: security, performance in software, performance in hardware, and flexibility. While security is commonly recognized as the most important evaluation criterion, it is also a measure that is most difficult to evaluate and quantify, especially during a relatively short period of time reserved for the contest. Therefore, performance evaluation plays a critical role in the selection and standardization of SHA-3. In this chapter, we present our proposed SHA-3 hardware evaluation methodology and ASIC implementation results for the SHA-3 fourteen Second Round and five Third Round candidates. Covering comprehensive hardware evaluation of all the SHA-3 Second and Third (Final) Round Candidates, the research described in this chapter aims to give a comprehensive overview of our consistent efforts in the SHA-3 hardware evaluation process.

4.1 SHA-3 ASIC Evaluation Methodology

Throughout the SHA-3 competition, NIST is looking for additional cryptanalytic results, as well as for performance evaluation data on hardware platforms. The SHA-3 submissions were made as a software reference implementation in combination with a set of test vectors. This pragmatic approach leverages ubiquitous computer infrastructure as a standard evaluation platform, and it suits the purpose of cryptanalysis. However, the reference implementations in C are also far away from actual hardware design. As a result, significant additional design work is required before the SHA-3 candidates can be evaluated in terms of hardware cost.

In contrast to software implementations, which can be characterized based on performance (execution time) only, hardware implementations have at least one additional dimension: resource cost, in addition to performance. Indeed, for hardware implementations, the architecture of the design represents an additional degree of design freedom. As a result, there is no single optimal hardware implementation. Every design has to be considered as a combination of performance under a given resource cost. This aspect complicates the comparison of designs. One may look for minimal resource cost under a given performance, or else for maximal performance under a given resource cost. Hence, a hardware benchmarking methodology needs to take this duality into account.

eBACS is a well known benchmarking environment, including a scripting environment and a performance database, for the evaluation of crypto-software [23]. Compared to the proposed methodology for benchmarking crypto-software, benchmarking crypto-hardware is ad-hoc. There are several reasons why the same progress is not seen in the hardware design community. All of them boil down to a lack of standardized approaches towards the design process.

First, there are no standard methodologies to quantify the cost and performance of a hardware implementation. In the average crypto-hardware conference proceedings, one will find that no two authors measure resource cost or performance of hardware implementations

using the same metrics. For example, the 11 tables that compare hardware implementations in the proceedings of CHES 2008 contain 18 different metrics for hardware cost and 10 different metrics for hardware performance [105]. While one author may use clock cycles, another one may use nanoseconds, and a third one blocks-per-second. It is up to the reader to provide the proper context.

A second reason is that hardware implementations show a larger heterogeneity compared to software processors. This includes the design target (ASIC or FPGA), the technology node, and the optimization scenario being used. Again, it is up to the reader to provide the proper context when making comparisons.

A third reason is the lack of standardized interface mechanisms for crypto-hardware modules. Because the architecture of a hardware design is a design decision, designers tend to count the interface as part of that freedom. This, however, significantly complicates benchmarking. Indeed, a standard Application Programming Interface (API) is a key enabler in existing software benchmarking environments such eSTREAM [42] and eBACS [23].

In this section we report on a methodology to address these issues for the SHA-3 ASIC benchmark process.

4.1.1 Overview

Figure 4.1 illustrates the overall design flow in our ASIC implementation. A set of RTL SHA-3 candidates is implemented in Verilog or VHDL. These hardware descriptions are next mapped to FPGA technology or ASIC technology. We use the same RTL descriptions for both types of design flow. Our objective is to use the FPGA as a prototyping technology for the ASIC, rather than a direct technology target. Hence, dedicated FPGA optimizations, such as the use of specialized multipliers or memory cells, are not used.

The ASIC and FPGA design flows look very similar, and cover the same two technology mapping steps. The first step is synthesis and mapping of the RTL code (in Verilog or

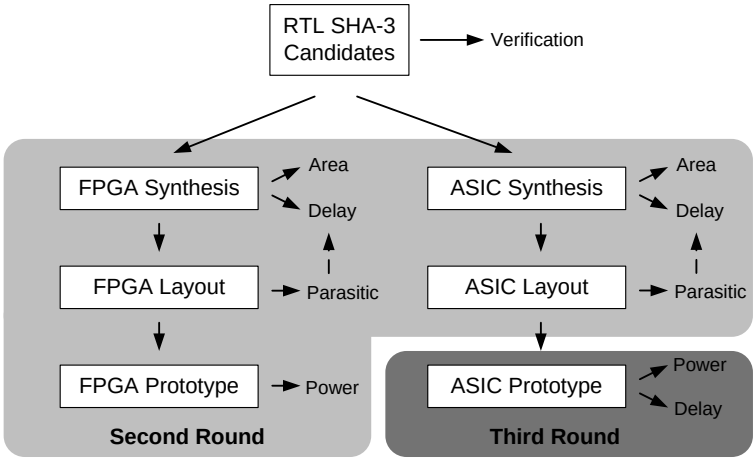


Figure 4.1: An overview of the SHA-3 ASIC evaluation project.

VHDL) to a netlist of technology primitives. The second step is place and route, and this step decides the spatial relationships of technology primitives in a layout. Both of these steps can be automated using scripts. The results of technology mapping are performance estimates such as circuit area and circuit delay. The performance delays obtained after place-and-route are more accurate than those obtained after synthesis. With respect to the circuit area, place-and-route will reveal the precise dimensions of the ASIC design. With respect to the circuit delay, place-and-route reveals implementation effects (annotated as parasitics in Fig. 4.1) which characterize delay effects caused by the interconnections.

The result of the ASIC and FPGA design flow is used in a prototype design based on the SASEBO board. In the case of ASIC design, we implement a chip with the five SHA-3 finalists.

4.1.2 Standard Hash Interface

When designing the standard hash interface for SHA-3 hardware, it is useful to refer to the method used to interface SHA-3 candidates in software. The software uses a NIST suggested Application Program Interface (API), and three function calls are used.

- `void init(hashstate *d)` initializes the state value of hash.
- `void update(hashstate *d, message *s)` hashes a message of a given length and updates the hash state. The message is chopped into pieces of a standard length called a message block (e.g. 256- or 512-bit). In case the message length is not an integral number of blocks, the API will use a *padding* procedure which extends the message until it reaches an integral number of blocks in length.
- `void finalize(hashstate *d, digest *t)` extracts the actual digest from the hash state.

A hardware interface for a SHA-3 module will emulate similar functionality as the software API interface. The hardware interface will therefore need to address the following issues.

Handshake protocol: The hash interface needs to synchronize data transfer between the SHA-3 module and the environment. This is done using a handshake protocol and one can distinguish a *master* protocol from a *slave* protocol, depending on which party takes the initiative to establish synchronization. The interfaces by Chen *et al.* [34] and Kobayashi *et al.* [89] use a slave protocol for the input and the output of the algorithm. The interfaces by Baldwin *et al.* [15] and Gaj *et al.* [52] define a slave protocol for the input and a master protocol for the output. The former type of interface is suitable for a coprocessor in an embedded platform, while the latter type of interface fits into the high-throughput applications that attach the SHA module to FIFO's. The key issue for a fair comparison is to use a common interface for all candidates. Therefore, we select the interface proposal of Chen *et al.* [34] (with a data I/O width of 16-bit), but observe that other proposals may be equally valid choices.

Wordlength: Typical block and digest lengths are wider (e.g. 512-bit) than the word length that can be provided by standard platforms (e.g. 32-bit), so that each hash operation will result in several data transfers. While this overhead is typically ignored by hardware designers, it is inherently part of the integration effort of the SHA-3 module. All of the inter-

face proposals leave the standard interface word length undefined, although they implicitly assume 32-bit. In this work, we use a 16-bit interface, which is constrained by the data bus between the control FPGA and the cryptographic FPGA/ASIC on SASEBO boards.

Control: The functions of the software API need to be translated to equivalent hardware control signals. One approach, followed by Gaj *et al.* [52], is to integrate this control as in-band data in the message stream with a pre-defined data structure. A second approach is to define additional control signals on the interface, for example to indicate message start and end. This is the approach taken by Chen *et al.* [34], Kobayashi *et al.* [89], and Baldwin *et al.* [15]. We follow the same approach in our work as well.

Padding: Finally, *padding* may or may not be included in the SHA-3 hardware module. In the latter case, the hardware module implicitly assumes that an integral number of blocks will be provided for each digest. Common padding schemes are defined by in-band data formatting, and this makes it possible to implement padding outside of the hardware module. The interface proposal by Baldwin *et al.* [15] explicitly places padding hardware into the interface. The other interface proposals leave padding to the SHA-3 designer. However, Chen *et al.* [34] and Kobayashi *et al.* [89] assume hardware padding will only be implemented at word-level, while Gaj *et al.* [52] supports bit-level padding as well. We follow the approach of Chen *et al.* [34] and Kobayashi *et al.* [89].

Note that there are many possible solutions to the interface problem, and that we present one possible approach. We observe that the key issue for a fair comparison is to use a *common* interface for all candidates. In addition, we will show that our performance evaluation mechanism allows to factor out the overhead of the interface communication.

4.1.3 Design Strategy

Besides a standard platform, our approach also defines a design strategy. There are three types of architectures available shown in Fig. 4.2, and we selected one of them for evaluation.

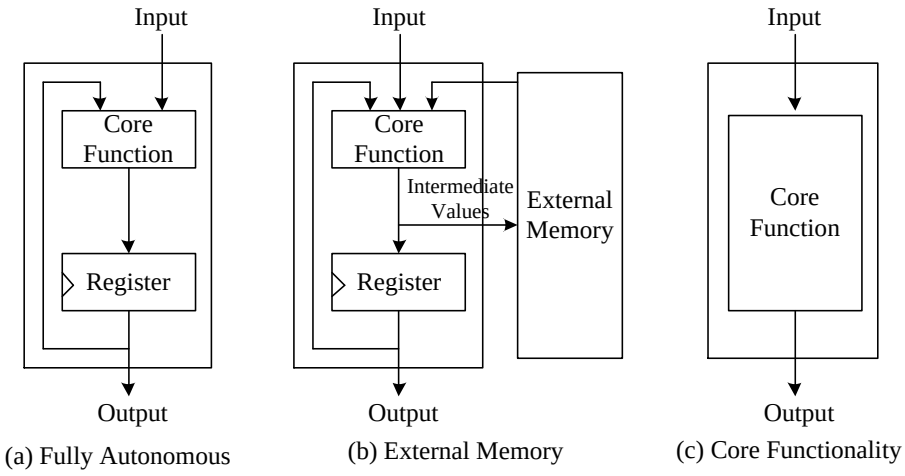


Figure 4.2: An overview of the three design strategies in hash function implementations.

Fully Autonomous In this architecture, one transfers message data to a hash function over multiple clock cycles, until a complete message block is provided. The hash module buffers a complete message block locally, before initiation the hash operation. Therefore, this architecture can work autonomously, and the resulting hash module is well suited for integration into other architectures.

External Memory In this architecture, only the data necessary for executing the hashing calculation are stored in registers. Other data (e.g. intermediate values) are stored in the external memory. This architecture can load the data from the external memory when it needs for hash calculation. The external memory is less expensive than registers in general. Therefore, the hash function hardware becomes a low-cost implementation. However, the architecture requires overhead cycles for accessing an external memory, and obviously it is not suitable for high-speed implementations.

Core Functionality This architecture has only the core part of a hash function, and ignores the storage of a full message block. In other words, this architecture assumes an "ideal" interface where the overhead of the data access is ignored in the hashing process.

The previous work for evaluating SHA-3 hardware performance did not use a *common* architecture. For example, the design method by Namin *et al.* [111] and Baldwin *et al.* [15] is based on the core functionality type and they evaluate a rough estimate of the performance of hash function hardware. On the other hand, the design method by Tillich *et al.* [129] is based on the fully-autonomous type. They assumed that the input data for the hash function hardware is sent in one cycle, so that the length of the input data is assumed long (e.g. 256- or 512-bit).

In this work, we evaluate the performance of SHA-3 candidates when they are used in a real system. In addition, we use performance metrics that enable us to separate the interface operation from the hash operation. On the cryptographic FPGA of SASEBO-GII and cryptographic ASIC of SASEBO-R [4], we all use the Fully Autonomous type. Compared to External Memory, this approach avoids off-chip memory access, and it also avoids the idealized (unrealistic) interface utilized by Core Functionality.

4.1.4 Platform for Integrated FPGA Prototyping and ASIC Performance Evaluation

The experimental environment for FPGA prototyping contains a PC, a SASEBO-GII [4] board and an oscilloscope. A SASEBO-GII board contains two FPGAs: a control FPGA, which supports the interfacing activities with a PC, and a cryptographic FPGA, which contains the hashing candidate. During the ASIC prototyping phase, the cryptographic FPGA is replaced by an ASIC containing SHA-3 candidates. A board from the SASEBO-R series will be used for this purpose.

The SASEBO board was originally developed for side-channel analysis. Hence, a potential research area for the FPGA prototype is side-channel analysis of SHA-3 candidates. In our experiments, we used the SASEBO board for a more obvious application, namely the measurement of power dissipation of the SHA-3 candidates mapped to the cryptographic

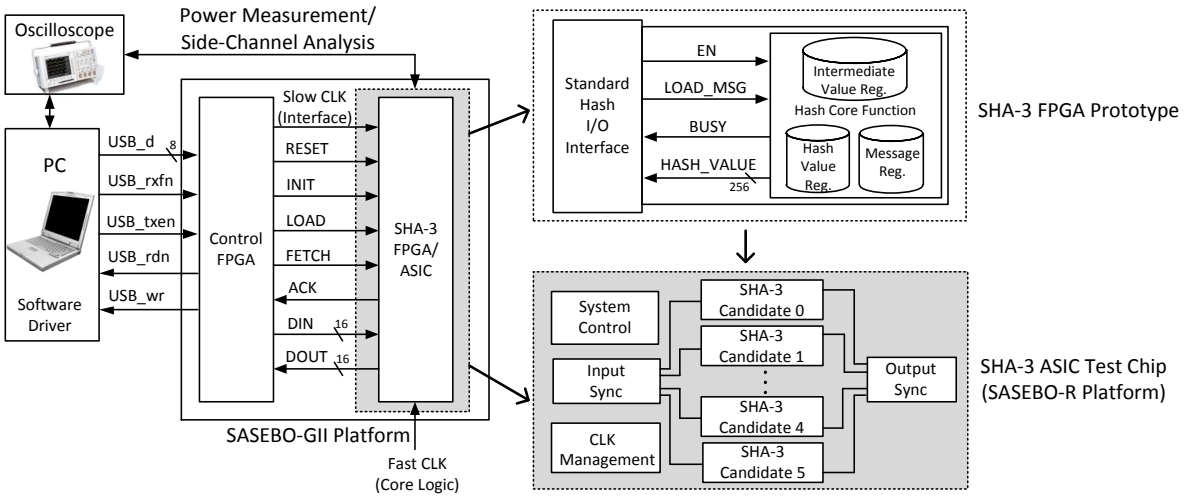


Figure 4.3: Experimental environment for FPGA prototyping and final ASIC testing.

FPGA.

The interface of the SASEBO board on the PC side is a software driver that can read the test vectors and that can send messages to the SHA-3 FPGA through USB. The Control FPGA manages the data flow of the messages and generates control signals according to the timing requirements of a standard hash interface. After SHA-3 FPGA finishes hash operations, the digest is returned to the PC through the Control FPGA. For the final ASIC prototype, the same data flow is used.

4.1.5 Optimization Target

The use of *Throughput-to-Area ratio* as the optimization target for SHA-3 hardware evaluation was first proposed by Gaj *et al.* [52], and later appeared in NIST Status Report on the Second Round of the SHA-3 Competition as a hardware evaluation criterion [133]. One of the obvious advantages of choosing this target rather than *Throughput* alone is that it can avoid highly unrolled hash designs with small throughput benefits but significant circuit area overhead.

4.1.6 Evaluation Metrics

In this work we have used *Throughput-to-Area ratio* as a primary metric and also reported other common metrics, including area, maximum frequency, maximum throughput, and power/energy efficiency.

Area We will use the circuit area of each SHA-3 candidate with both the interface and hash core after layout. The area will be reported in kilo gate equivalents (kGE), where a gate equivalent corresponds to the area of a standard NAND2 gate in the standard-cell library. We divided the reported layout area in μm^2 by the area of an NAND2 gate, for conversion from the absolute circuit area to kGE.

Throughput In general the time required to hash a message consists four parts: the latency for loading one block of message, L_{in} , the hash core latency, L_{core}, the latency for finalization step, L_{final}, and the latency for outputting the message digest, L_{out}. For short message hashing, all these four latencies are important performance factors. The metric of *Latency* is frequently used to characterize the short message hashing speed instead of *Throughput*. In the case of hashing a long message, L_{final} and L_{out} can be neglected. Since L_{in} is dependent on the system I/O throughput which may vary in different contexts, here we report the *Throughput* results of the hash core function:

$$Tp_{core} = \frac{W_B \times f_{max}}{L_{core}}, \tag{4.1}$$

where W_B is the block size of the hash and f_{max} is the maximum frequency the circuit can run at.

Throughput-to-Area The *Throughput-to-Area Ratio* is an effective metric to measure the hardware efficiency, where the *Throughput* is the above defined Tp_{core} and the *Area* is

for the layout circuit area expressed in terms of kGE.

Power/Energy The power is measured with a fixed achievable clock frequency based on the average power during hashing of long messages, and the capture period is only for the core hashing operations (e.g. round function for each message block). The energy metric is expressed as energy per bit of the input message block compressed by the hash core function.

4.2 VLSI Implementation of SHA-3 Hash Functions

Although we have fixed our optimization target, *Throughput-to-Area ratio*, to fully understand each SHA-3 candidate by its specification and reference C codes and optimize its hardware implementation to achieve the goal is far from trivial. We had looked into several reference implementations [3,24,64] and optimized them for our system architecture. In this section, we describe the VLSI architecture of all the five SHA-3 finalists. For the detailed algorithm descriptions, please refer to the official NIST SHA-3 website [112].

4.2.1 BLAKE

BLAKE follows a HAIFA iteration mode and operates on an inner state that can be represented as a 4 by 4 matrix of words. The inner state is initialized using an Initial Value (IV), a salt and a counter. The state is updated using the **G** function, which is based on the ChaCha stream cipher [18]. The **G** function mainly contains modular addition, XOR, and rotate operations.

As shown in Fig. 4.4, before the message block enters into the round function it will first go through a permutation layer and XOR with predefined constants. One stage of pipeline is inserted inside the permutation layer for higher throughput. Four parallel **G** functions are implemented for the round process. Eight **G** functions or fully unrolled structures are

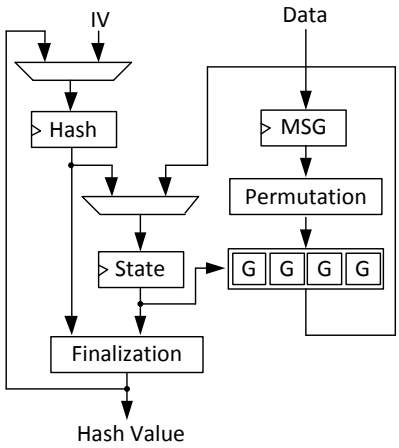


Figure 4.4: Structure of the BLAKE-256 core.

also possible high performance solutions if there is no area constraint. Each **G** function instantiates two carry-save adders.

4.2.2 Grøstl

Grøtl is a wide-pipe Merkle-Dåmgard hash algorithm with an output transformation. The compression function is a novel construction, using two fixed $2n$-bit permutations together. The output transformation processes the final chaining state, and discards half the bits of the result, yielding an n-bit hash output. The underlying fixed permutations are themselves based closely on the structure of AES, reusing the S-box, but expanding the size of the block to 512-bit for Grøtl-256 in a straightforward way.

Grøstl can be implemented with parallel computation of **P** and **Q** permutations as well as a single permutation with iterations. As shown in Fig. 4.5, we implement the parallel **P** and **Q** structure, which enables better *Throughput-to-Area ratio* in our case since the hash core throughput can be doubled without doubling the overall area when considering the hardware interface and control circuits overhead. Within the parallel **P** and **Q** structure, 128 AES SBoxes are used and all of them are implemented in Galois field operations also for better hardware efficiency.

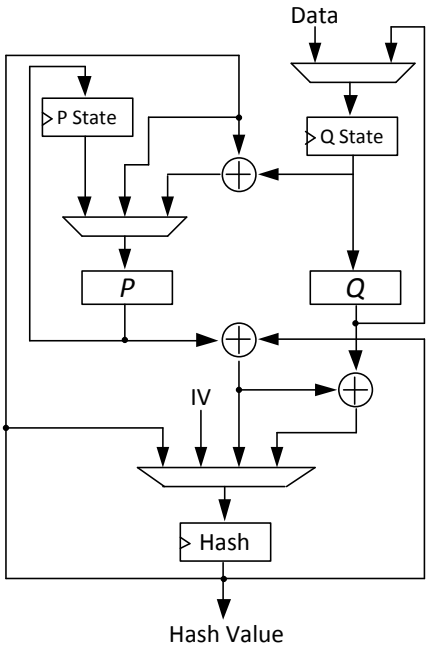


Figure 4.5: Structure of the Grøstl-256 core.

4.2.3 JH

The compression function of JH is constructed from a large block cipher with constant key. The large block cipher is based on a generalized AES design methodology and can be constructed with small components. There are three components of the underlying permutation **E8** : 4-bit SBoxes, an 8-bit L-permutation, and a P-permutation.

As shown in Fig. 4.6, two similar round operations, **R8** and **R6**, are used for compression and round constant generation, respectively. **R8** is used to update the 1024-bit hash state, and **R6** generates the round constant on-the-fly.

4.2.4 Keccak

Keccak follows the sponge construction model. The permutation can be considered as a substitution-permutation network with 5-bit wide SBoxes, or as a combination of a linear mixing operation and a very simple nonlinear mixing operation. The building block permu-

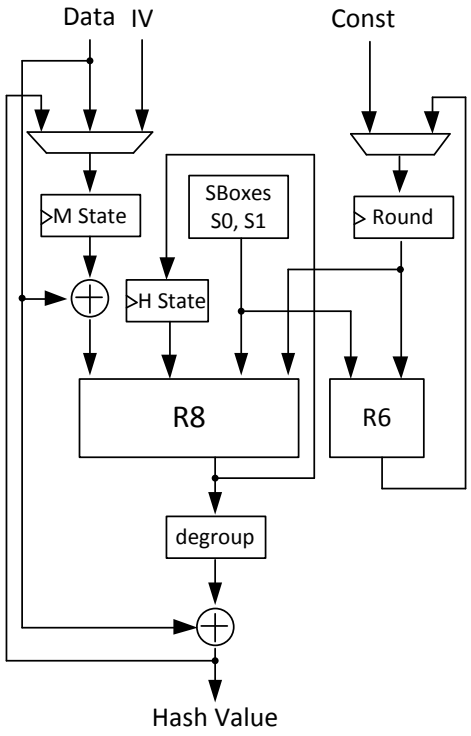


Figure 4.6: Structure of the JH-256 core.

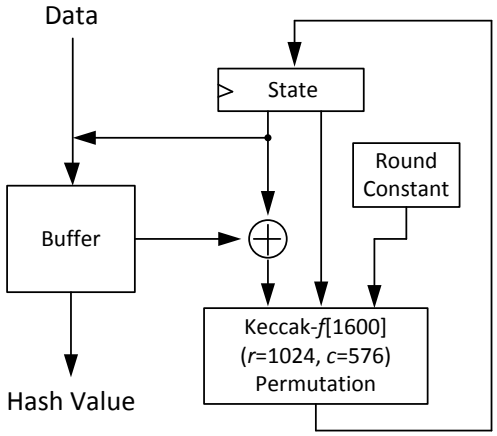


Figure 4.7: Structure of the Keccak-256 core.

tation is from a set of 7 permutations, indicated by Keccak-$f[b]$ (b is the width of Keccak-f with default value 1600).

We implement the Keccak-$f[1600]$ with rate $r = 1024$, capacity $c = 576$, and output

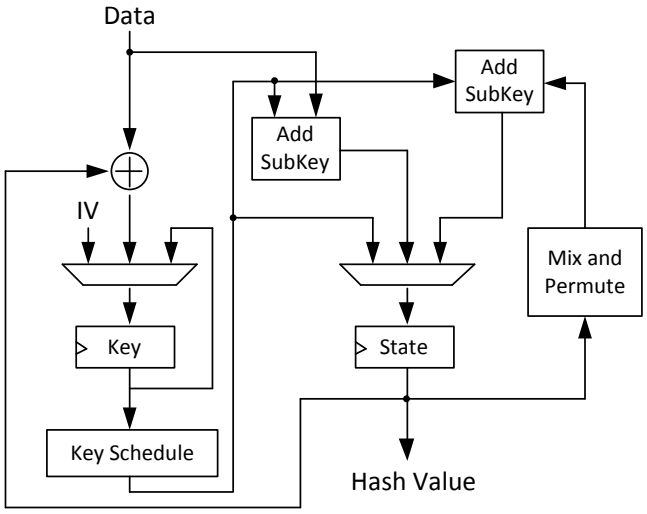


Figure 4.8: Structure of the Skein512-256 core.

digest size of 256-bit. The Keccak core design are based on the authors' provided reference high speed core designs [24].

4.2.5 Skein

Skein is an iterative hash algorithm built on a tweakable block cipher C Threefish. Threefish is used to build the compression function of Skein using a modified Mateas-Meyer-Oseas construction, which is then iterated in a chaining mode similar to HAIFA. The designers refer to the whole construction as a Unique Block Iteration (UBI) mode. Threefish is a 72-round substitution-permutation network using a 128-bit MIX function consisting of a 64-bit addition, rotate and XOR operations.

As shown in Fig. 4.8, we implemented the Skein512-256 version and each Threefish-512 round out of 72 rounds consists of four parallel MIX operation and a permutation. Four rounds are unrolled and chained together in hardware.

4.3 SHA-3 Round 2: ASIC Evaluation of Fourteen Second Round Candidates

In this section, we compare the SHA-3 ASIC results for the SHA-3 fourteen Second Round Candidates, and we address the impact of different factors that are quite relevant for fair and comprehensive evaluation. These factors include technology differences, ASIC layout overhead over the post-synthesis results, various application-specific constraints, and different hash operation modes.

4.3.1 ASIC Design Parameters

The performance evaluation of a design in ASIC technology can be done under multiple technologies. Rather than evaluating all 14 candidates under multiple technologies, we first evaluate a single candidate under different ASIC design parameters as follows.

- We evaluate the impact of different technologies. A smaller technology is smaller and faster, but may also have increased static power dissipation.
- We evaluate the impact of different constraints. During technology mapping, a given RTL design can be optimized for area, speed, or a combination of those.
- We compare Post-Synthesis results *vs.* Post-Layout results. ASIC layout provides additional implementation characteristics such as precise area and netlist parasitics.
- We evaluate the impact of message length. Because the regular processing, and the final processing of a hash candidate can differ, the message length may affect the average activity of a hash implementation. This will affect the power dissipation.

To evaluate these parameters, we used the Synopsys Design Compiler (C-2009.06-SP3) to map the CubeHash RTL codes to UMC 90nm (FSD0A_ A_ GENERIC_ CORE_ 1D0V_

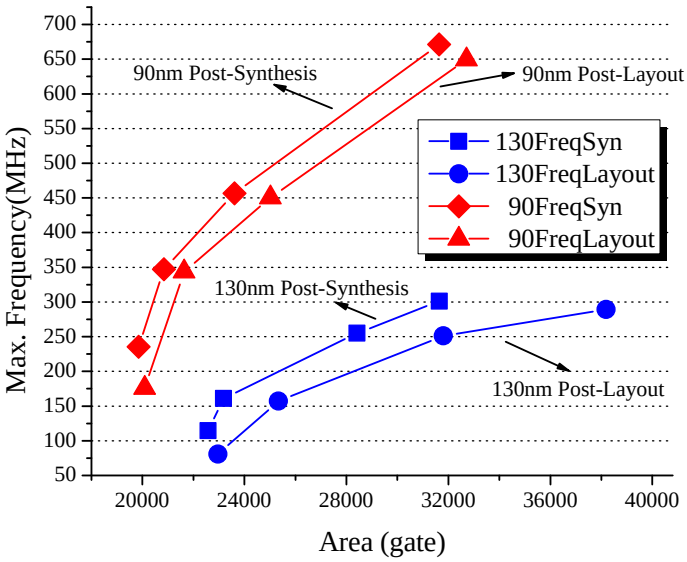


Figure 4.9: Area and speed results of ASIC implementation of CubeHash-256.

TP_ 2007Q1v1.7) and 130nm (FSC0G_ D_ SC_ TP_ 2006Q1v2.0) technologies. We use the typical case condition characterization of the standard cell libraries. The 90nm technology uses 9 metal layers, and the 130nm technology uses 8 metal layers. In general, more metal layers allow for a denser interconnect, and hence a more optimal use of die area.

1. *MinArea*: A minimum-area design will minimize the use of logic resources (gates) at the expense of performance.

2. *MaxSpeed*: A maximum-speed design will minimize the computational delay of the design, at the expense of area.

3. *TradeOff0*: The first trade-off point is chosen to have a computational delay which is two-thirds between the MinArea and MaxSpeed design points.

4. *TradeOff1*: The second trade-off point is chosen to have a computational delay which is five-sixths between the MinArea and MaxSpeed design points.

The TradeOff points are chosen to investigate how the relationship (speed, area) evolves when a design gradually moves from the MinArea design point to the MaxSpeed design

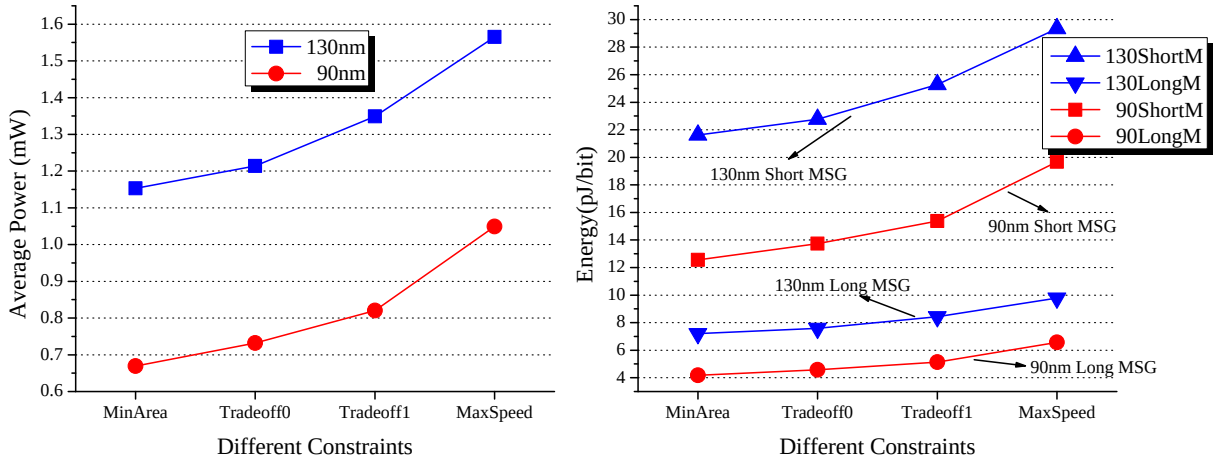


Figure 4.10: Power and energy results of ASIC implementation of CubeHash-256.

point.

The Synopsys IC Compiler (C-2009.06-SP5) is used for the back-end process. For all the designs we start with 85% utilization of the core area. The utilization is defined as the die area devoted to active components (standard cells) as compared to the total die area. Due to the routing of signals, power, and ground between active components, utilization can never reach 100%. The optimal value for utilization should be as high as possible. After place-and-route, design flow errors such as timing and Design Rule Check (DRC) violations may occur. In that case, the initial utilization must be lowered in order to relax the constraints to the place-and-route process.

The timing results can be obtained from the post-synthesis and post-layout steps. First, the Synopsys IC Compiler is used to extract the post-layout parasitic and generate an SDF file containing the delays of all the interconnections and instances. Second, Synopsys VCS can be used to do the post-simulation and generate the VCD file that records all the switching activities of the netlist. Finally, Synopsys Prime Time (C-2009.06-SP3) reads the final netlist, VCD file and .spef parasitic file and does the power estimation.

Fig. 4.9 and Fig. 4.10 show the results of these technology parameters on the implementation of the CubeHash-256 SHA-3 candidate. Fig. 4.9 is an area-delay plot, which marks

the area of a given design against the achievable performance (in this case, the maximum clock frequency). The X-axis of Fig. 3 is calibrated in equivalent gates. This means that the area is normalized to a standard 2-input NAND gate in the chosen technology. Fig. 4.10 is the power and energy plot that illustrates the impact of different design optimization constraints, technology, and message characteristics. The left pane of Fig. 4.10 indicates the average power dissipation during the processing of a very long message. The right pane of Fig. 4.10 indicates the energy dissipation per bit during the processing of messages of variable length.

- *The impact of different technologies.* The relationship between 130nm and 90nm technologies, as shown in Fig. 4.9, is non-trivial. However, one can notice that the relative relationship between the four points on each curve is similar. This means that a characterization in a single technology can also serve as a characterization in nearby technology nodes. In our experiments, we concentrated on area-delay characterization in 130nm technology.

- *The impact of different constraints.* As illustrated in Fig. 4.9, the impact of constraints (MinArea, MaxSpeed, TradeOff0, TradeOff1) is significant, and it varies the performance by a factor of almost 3. In exploring the 14 SHA-3 candidates, we have therefore fully characterized the 4 design points of each design in 130nm technology.

- *Post-Synthesis results vs. Post-Layout results.* Fig. 4.9 illustrates obvious differences between post-synthesis and post-layout results. Because post-synthesis results provide higher accuracy, we have obtained post place-and-route results for all 14 SHA-3 candidates.

- *The impact of message length.* From the energy results shown in Fig. 4.10, we can clearly see that energy per message bit changes a lot when considering different message lengths. Note that the power consumption is the same for CubeHash message update step and finalization step since those two steps calls the same round functions with

different rounds. The cause of the energy differences is due to the different throughput and latencies for short and long messages.

4.3.2 Analysis of Post-Layout Results

In this section we present the performance results of the SHA-3 ASIC implementations with the UMC 130nm standard cell technology. Design space exploration is performed for all the 14 second round candidates. For each of the graphs shown below there will be 4 points on the curve representing the Min Area, Max Speed and two tradeoffs points.

In Fig. 4.11, the throughput is calculated based on the maximum clock frequency of the post-layout design and only consider hashing long messages. We also report the results for short and long message cases in Table 4.1.

Fig. 4.11 illustrates how architecture differences affect the performance results. Some curves, like those of Keccak and Luffa, are very steep. This means that a small increase in area yields a significant performance improvement. Other curves however are relatively flat. For a design such as SIMD, for example, even a large addition of gates will not yield additional performance. The optimal points in Fig. 4.11 are those with maximal performance and minimum area. This optimum is located on the upper left side of the graph. The curves of Keccak and Luffa are clearly out-shadowing other designs.

To compare the results of the SHA-3 candidates, we use the methodology proposed by Gaj *et al.* [52]. Therefore, we utilize a uniform metric, Throughput-to-Area Ratio, as the primary metric to rank all the designs. The SHA-3 design with higher Throughput-to-Area ratio means with given fixed hardware resources this SHA-3 candidate has better efficiency (hash more message in the same period of time).

Fig. 4.12 shows the Throughput-to-Area ratio graph for all the 14 SHA-3 candidates. We can also observe how this 'efficiency' metric changes according to different constraints. By looking at the results shown in Fig. 4.12, if only considering the 'Throughput-to-Area ratio'

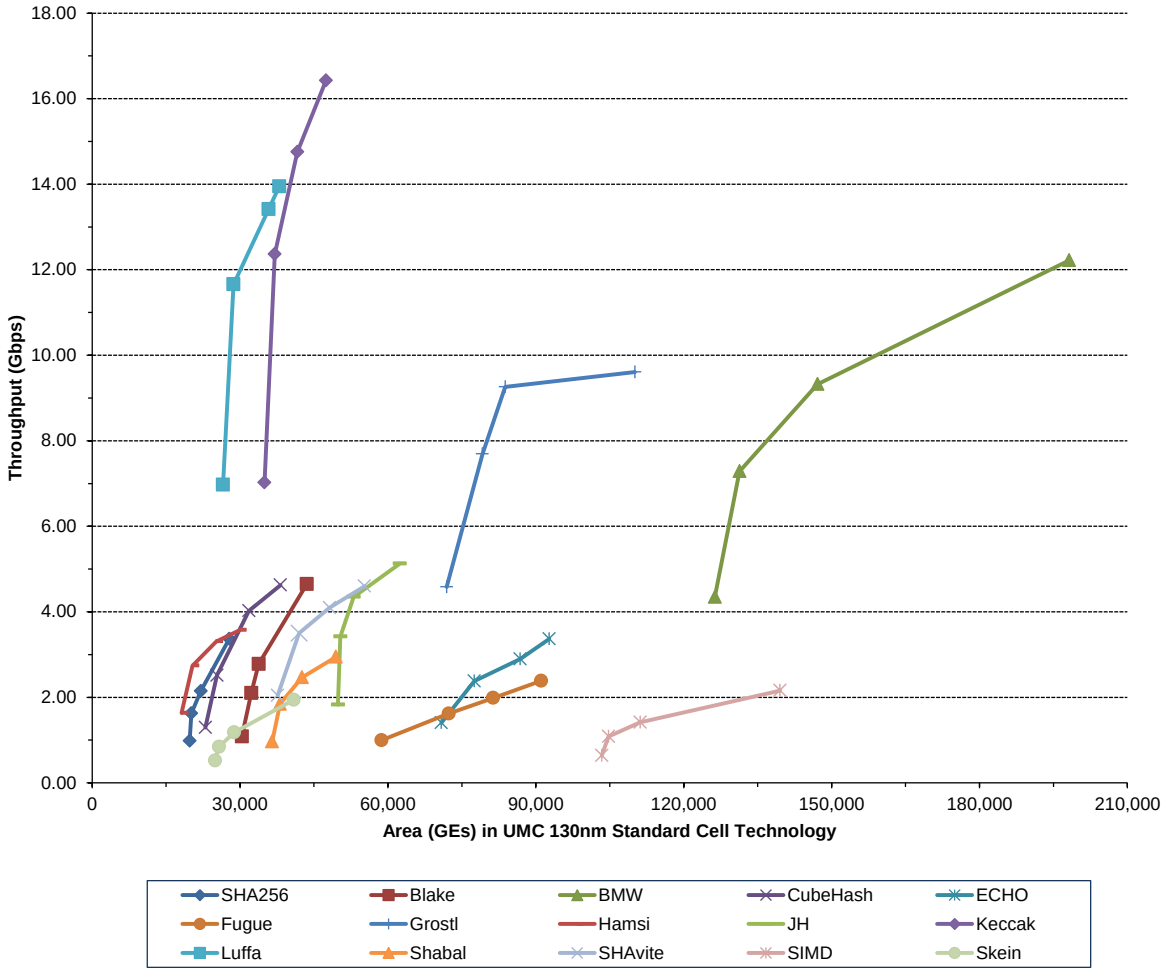


Figure 4.11: Post-Layout results for throughput and gate counts in UMC 130nm standard cell technology.

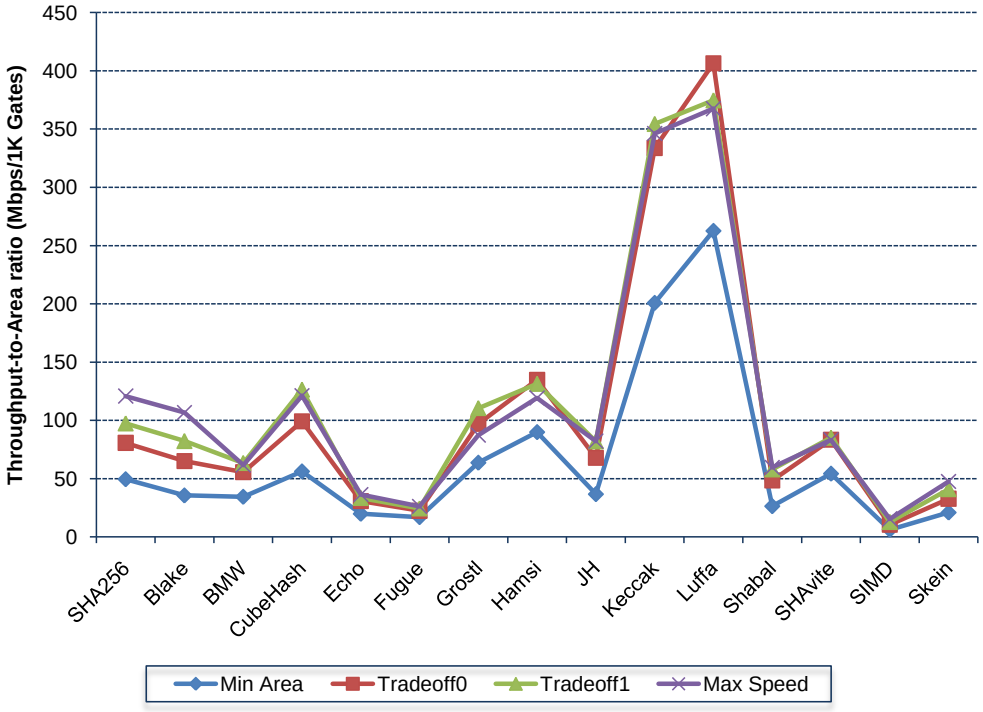


Figure 4.12: Throughput-to-Area ratio for all the designs with 4 different constraints.

metric, the ranking of the 14 SHA-3 designs can be found in Table 4.1. The SHA-256 is also included to serve as a reference.

Although it is not necessary that the new SHA-3 standard has to be better than the existing SHA-256 in terms of performance, still one would be interesting to see the comparison results. In Fig. 4.13, for all the 4 cases, the Throughput-to-Area ratio of all the designs has been normalized to the value of SHA-256. All the points that are above the red line, which denotes value one, are outperforming the SHA-256.

For detailed analysis we have shown all the results in Table 4.2 with some notes:

1. 'I/F+Core' cycle counts is equal to $I_{in} + I_{core}(I)$.

2. 'Core' cycle counts is equal to $I_{core}(I_{core} + I_{final})$.

3. LongMSG and ShortMSG cases include the communication overhead by interface.

Table 4.1: Ranking of the 14 SHA-3 designs in terms of Throughput-to-Area ratio metric

Rank	*MinArea*	*Tradeoff 0*	*Tradeoff 1*	*MaxSpeed*
1	Luffa	Luffa	Luffa	Luffa
2	Keccak	Keccak	Keccak	Keccak
3	Hamsi	Hamsi	Hamsi	CubeHash
4	Grøstl	CubeHash	CubeHash	SHA256
5	CubeHash	Grøstl	Grøstl	Hamsi
6	SHAvite	SHAvite	SHA256	BLAKE
7	SHA256	SHA256	SHAvite	Grøstl
8	JH	JH	BLAKE	SHAvite
9	BLAKE	BLAKE	JH	JH
10	BMW	BMW	BMW	BMW
11	Shabal	Shabal	Shabal	Shabal
12	Skein	Skein	Skein	Skein
13	Echo	Echo	Echo	Echo
14	Fugue	Fugue	Fugue	Fugue
15	SIMD	SIMD	SIMD	SIMD

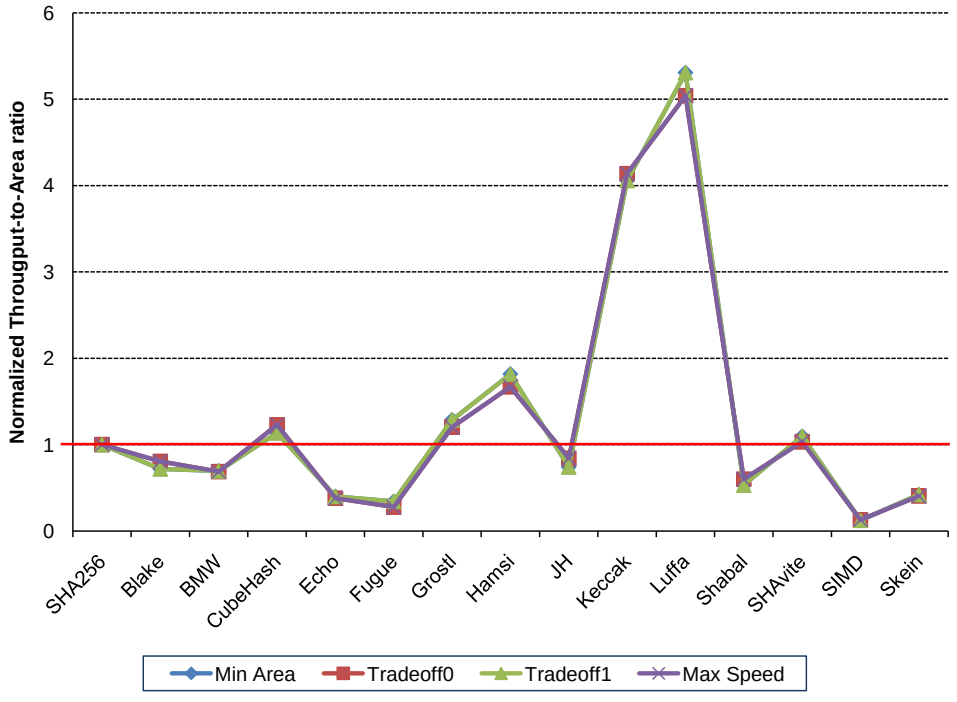


Figure 4.13: Normalized Throughput-to-Area ratio for all the designs with 4 different constraints.

4. The values in parenthesis are the case excluding the interface overhead, e.g. only the core function block.

Table 4.2: Performance results of post-layout designs of the SHA-3 14 candidates with UMC 130nm technology

		Block	Max	# of cycles		LongMSG	ShortMSG	Area
		Size	Freq.	IF+Core	Core	TP[Mbps]	Latency[us]	[Gates]
SHA256	MinA	512	130	148(196)	68(68)	450(979)	3.81(1.57)	19789
	MaxS	512	446	148(196)	68(68)	1544(3361)	1.11(0.46)	27816
BLAKE	MinA	512	46	121(169)	22(22)	196(1080)	8.92(1.42)	30365
	MaxS	512	200	121(169)	22(22)	845(4645)	2.07(0.33)	43521
BMW	MinA	512	17	98(148)	2(4)	89(4345)	20.27(0.35)	126315
	MaxS	512	48	98(148)	2(4)	249(12220)	7.21(0.13)	198167
CubeHash	MinA	256	81	64(272)	16(176)	323(1290)	6.58(2.98)	22968
	MaxS	256	289	64(272)	16(176)	1156(4624)	1.84(0.83)	38184
ECHO	MinA	1536	90	407(455)	99(99)	342(1404)	5.06(1.09)	70850
	MaxS	1536	217	407(455)	99(99)	819(3366)	2.11(0.46)	92727
Fugue	MinA	32	62	8(93)	2(39)	249(995)	5.61(1.66)	58705
	MaxS	32	149	8(93)	2(39)	596(2385)	2.34(0.69)	91089
Grøstl	MinA	512	89	106(164)	10(20)	432(4580)	4.24(0.45)	71933
	MaxS	512	188	106(164)	10(20)	906(9606)	2.00(0.21)	110108
Hamsi	MinA	32	204	10(63)	4(9)	653(1633)	1.96(0.70)	18159
	MaxS	32	446	10(63)	4(9)	1429(3571)	0.90(0.32)	29941
JH	MinA	512	139	135(183)	39(39)	512(1828)	3.25(0.84)	49871
	MaxS	512	391	135(183)	39(39)	1481(5128)	1.16(0.30)	62417
Keccak	MinA	1024	161	217(265)	25(25)	761(6606)	2.99(0.31)	34959
	MaxS	1024	377	217(265)	25(25)	1781(15457)	1.28(0.13)	47434
Luffa	MinA	256	245	57(114)	9(18)	1101(6972)	1.41(0.22)	26551
	MaxS	256	490	57(114)	9(18)	2202(13943)	0.70(0.11)	37942
Shabal	MinA	512	118	143(341)	50(200)	424(962)	5.33(2.87)	36516
	MaxS	512	362	143(341)	50(200)	1297(2945)	1.74(0.94)	49439
SHAvite	MinA	512	152	134(185)	38(38)	579(2041)	2.97(0.55)	37621
	MaxS	512	341	134(185)	38(38)	1304(4599)	1.32(0.33)	55245
SIMD	MinA	512	57	142(190)	46(46)	206(636)	8.30(2.42)	103379
	MaxS	512	194	142(190)	46(46)	699(2157)	2.45(0.71)	139547
Skein	MinA	256	43	75(143)	21(41)	146(521)	8.67(2.43)	24919
	MaxS	256	159	75(143)	21(41)	544(1941)	2.33(0.65)	40899

4.3.3 Summary

In this section, we presented performance evaluation results of 14 SHA-3 Second Round candidates in a 130nm CMOS ASIC Technology. We discussed the impacts of various factors including technology, design constraints, place-and-route, and hash operating modes. We conclude that top-performing candidates in our experiment include Luffa, Keccak, Hamsi, Cubehash, and Grøstl.

4.4 SHA-3 Round 3: Design and Benchmarking of an ASIC with Five Finalists

In December 2010, the SHA-3 competition entered into Phase III and five SHA-3 candidates were selected for further evaluation as SHA-3 finalists. Although security is of primary importance, the lack of systematic cryptanalysis makes it very hard to compare the security strength of different hash candidates. Within the limited one year period for the final round evaluation, the cost and performance of SHA-3 software and hardware implementations aspects are expected to put more weight in the selection of SHA-3 winner.

For SHA-3 final round ASIC evaluation, we design a SHA-3 ASIC by following a fair and consistent SHA-3 hardware evaluation methodology [60, 65]. We start by defining a standard interface, and optimized the designs with a single metric, *Throughput-to-Area ratio*. Next, we developed an FPGA prototype that can provide a seamless transition into ASIC implementation. Finally, we designed an ASIC chip with all the five finalists using the latest Round 3 tweaks and SHA256 as a reference design.

The remainder of this section is organized as follows. Section 4.4.1 gives an overview of the SHA-3 ASIC benchmark status. In Section 4.4.2, the VLSI architecture of the SHA-3 ASIC will be described. The VLSI implementation of each SHA-3 finalist is discussed in Section 4.2. The silicon implementation constraints will be discussed in Section 4.4.3.

Section 4.4.4 describes the testing environment and presents the analysis of chip measurement results. Section 4.4.5 concludes the SHA-3 ASIC design.

4.4.1 Related Work

The hardware evaluation of SHA-3 candidates has started shortly after the specifications of 51 algorithms submitted to the contest became available. More comprehensive efforts became feasible only after NIST's announcement of 14 candidates qualified to the second round of the competition in July 2009. Since then, several comprehensive studies in SHA-3 ASIC implementations have been reported [60, 61, 73, 74, 88, 111, 129, 130]. Guo *et al.* [60] used a consistent and systematic approach to move the SHA-3 hardware benchmark process from the FPGA prototyping by Kobayashi *et al.* [89] to ASIC implementations based $130nm$ CMOS standard cell technology. Tillich *et al.* [129] presented the first ASIC post-synthesis results using $180nm$ CMOS standard cell technology with high throughput as the optimization goal and further provided post-layout results [130]. Henzen *et al.* [74] implemented several architectures in a $90nm$ CMOS standard cell technology, targeting high- and moderate-speed constraints separately, and presented a complete benchmark of post-layout results. Knezevic *et al.* [88] provided ASIC synthesis results in a $90nm$ CMOS standard cell technology as a comparison with their primary FPGA prototyping results. In December 2010, five candidates were selected for the last round of SHA-3 competition. These candidates then submitted the final specification of their algorithms in January 2011. The only comparison of the five candidates in ASIC implementations at this stage was provided by [65, 66] based on post-layout simulation.

Table 4.3: The related SHA-3 hardware benchmarking work in ASICs.

	14 Second Round Candidates				5 Third Round Finalists
	Tillich [129,130]	Guo [60]	Henzen [74]	Knezevic [88]	Guo [65,66]
Technology Choices	180nm CMOS	130nm CMOS	90nm CMOS	90nm CMOS	130nm CMOS
Hardware Interface	Assume infinite bandwidth interface	Defined standard 'handshake' interface	Assume infinite bandwidth interface	Defined standard 'handshake' interface	Defined standard 'handshake' interface
Chosen Metrics	Area, Throughput	Area, Throughput, Power, Energy	Area, Throughput, Energy	Area, Throughput, Power, Energy	Area, Throughput, Power, Energy
Design Flow	Post-layout/synthesis	Post-layout	Post-layout	Post-synthesis	Post-layout

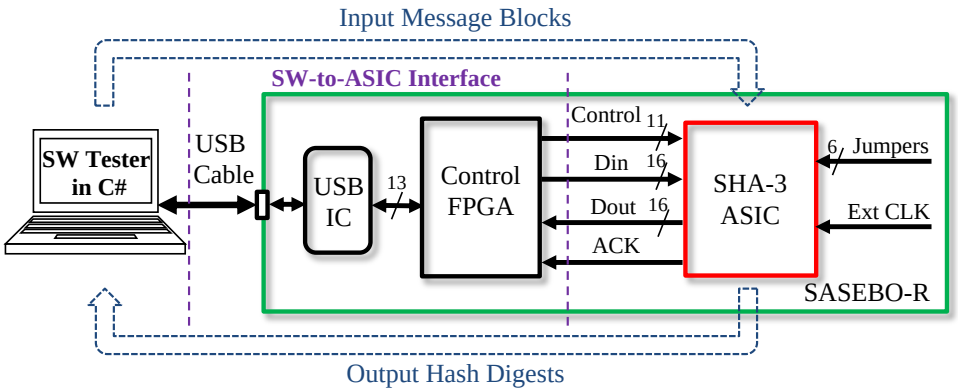


Figure 4.14: The SASEBO-R platform for SHA-3 ASIC testing.

4.4.2 VLSI Architecture of SHA-3 ASIC

This section covers the VLSI architecture design of the SHA-3 ASIC as shown in Fig. 4.15 under the physical constraints imposed by the selected SASEBO-R board [4]. As an open platform the SASEBO-R board was originally developed for side-channel analysis. Hence, a potential research area for SHA-3 ASIC is side-channel analysis of SHA-3 candidates. In our experiments, we used the SASEBO-R board for a more obvious application, namely the measurement of power dissipation of the SHA-3 candidates mapped to ASICs.

The experimental environment for SHA-3 ASIC contains a PC and a SASEBO-R board as shown in Fig. 4.14. A SASEBO-R board contains a control FPGA, which supports the interfacing activities with the PC and SHA-3 ASIC.

Clock Management

For signal integrity issues associated with the on-board wire transfers, the ASIC interface clock used to synchronize all the data and control signals from/to the control FPGA should only run at a relatively low frequency. The SHA-3 ASIC chip can operate at 250 *MHz* for some candidates, so an additional stable fast clock is required, which will be gated and shared by all the hash modules.

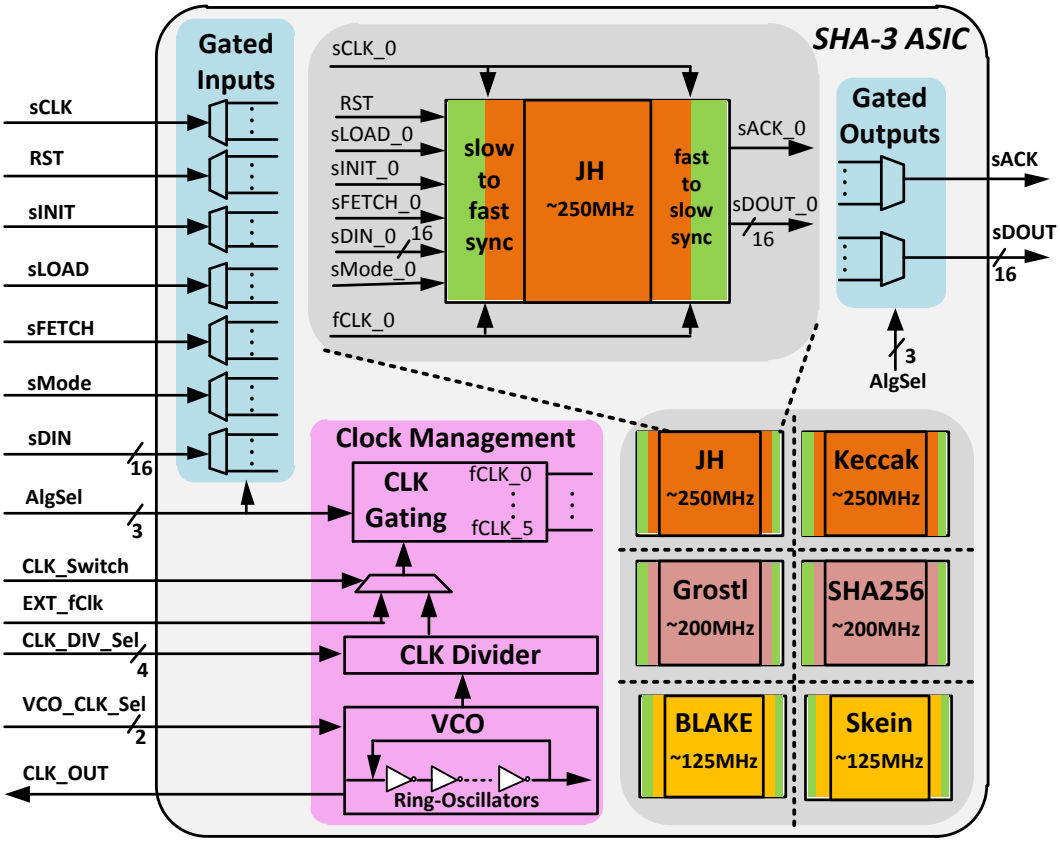


Figure 4.15: The block diagram of SHA-3 ASIC.

• *Clock Generation.* For high frequency testing purpose, an on-chip clock generation module is integrated. We used the custom-cell design approach to integrate a ring oscillator (RO) based voltage-controlled oscillator (VCO) into the chip. VCO is an electronic oscillator designed to be controlled in oscillation frequency by a DC voltage input(i.e. PBIAS port in SHA-3 chip). In addition, we also integrated three standard-cell ROs to provide fixed high frequency clocks.

The VCO takes four input ports PBIAS, VCO-EN-7,9,11, and three output frequencies, VCO-RO-7,9,11. The PBIAS voltage can be varied to produce a range of clock frequencies. The voltage can be varied from 0V to 0.8V. The 'EN' is used to turn on/off the clock outputs of VCO. VCO-RO-7,9,11 are the three frequencies produced by the block for any particular PBIAS voltage. VCO-RO-7 is the clock from a 7-stage RO.

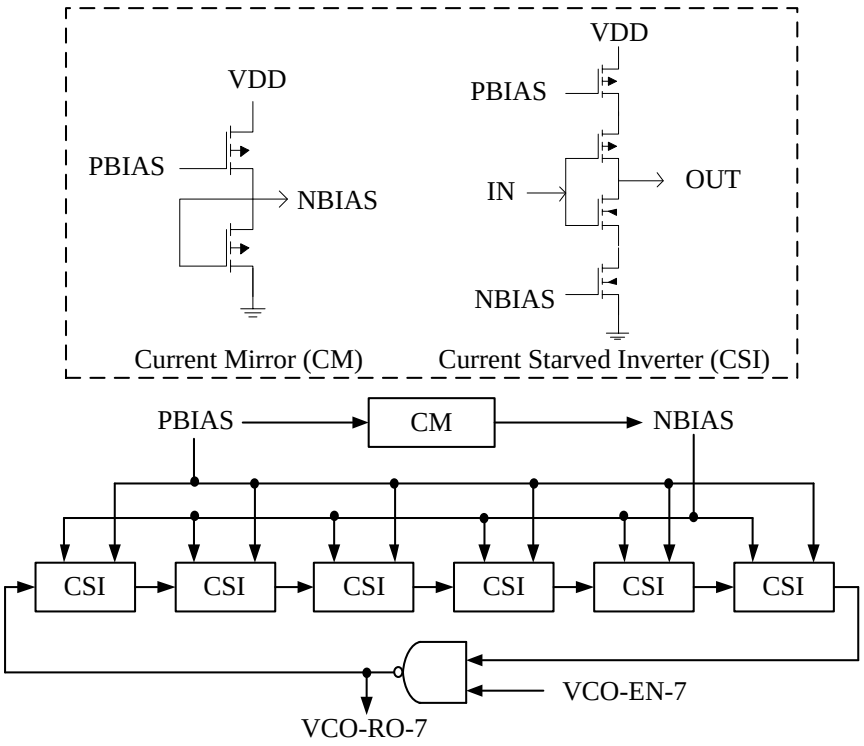


Figure 4.16: The diagram of a 7-stage VCO-RO clock design.

We did extensive Spice simulations to determine the optimum stage length and appropriate device dimensions. We performed simulations at different process corners, and we selected VCO configurations to meet the frequency requirements even in the worst process corners. Simulations have shown that the ROs with stage lengths of 7, 9, and 11 are required to generate the range of frequencies. As shown in Fig. 4.16, the Ring Oscillator (RO) of the VCO is realized using NAND2, Current Starved Inverter (CSI), and Current Mirror (CM) custom cells. We designed the custom cells using Cadence Virtuoso, and we performed the Design Rule Check and Layout vs. Schematic with Assura. We used Synopsys library compiler and Milkyway to generate libraries for synthesis and place and route. We used Synopsys Design Compiler (C-2009.06-SP3) and IC Compiler (C-2009.06-SP5) for synthesis and place & route, respectively. The RC parasitics have been extracted for the post-layout design and spice simulations have been performed for verification.

The standard cell RO-based clock generation together with the VCO clocks and a standard

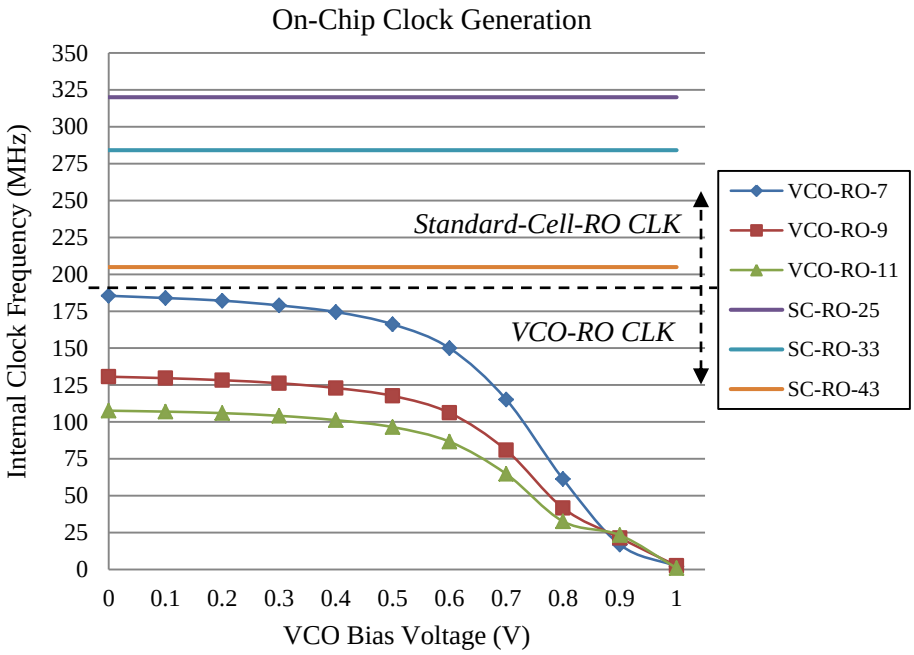


Figure 4.17: The range of on-chip generated clock frequencies.

cell implemented clock divider can support a wide range of clock frequencies to fill our need for performance testing. Fig. 4.17 shows averaged measurements of on-chip clock speed for a batch of 10 fabricated chips.

- *Clock Configurations.* The on-chip generated clocks are also MUXed with the external fast clock input and can be configured through dedicated ports. The external fast clock can be fed through a SMA connector from a signal generator or it can be provided through the control FPGA. Clock gating is implemented to guarantee that only one hash module is enabled at a time.

Chip Interface

- *Standard Hash Interface.* The chip interface shown in Fig. 4.15 adopted the standard hash interface by Chen et al. [34] and extended it to add mode selection and dual-clock support.

- *Clock Domain Crossing (CDC) Synchronizer.* There are two clock domains in our chip:

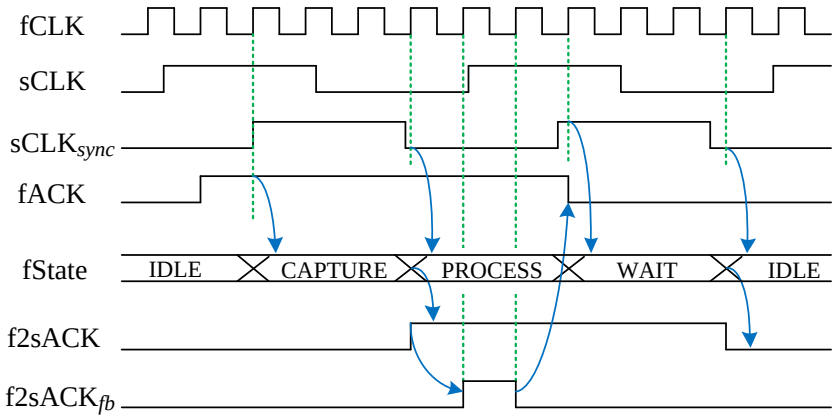


Figure 4.18: The timing of slow-to-fast synchronizer design.

the slow one is for the interfacing logic and the fast one is for hash modules. In order to avoid complex synchronizer designs based on asynchronous FIFOs or feedback synchronization and alleviate the burden of the backend process to deal with the two clock domains, we simplified the synchronizer design by making a reasonable assumption that the internal hash clock working frequency is always at least two times faster than the slow interface clock. As a result, the slow interface clock is treated as a plain control signal and the whole chip only has one single fast clock. To synchronize the slow interface signals to the fast hash core, a synchronizer with 2-stage flip-flops is used. The fast-to-slow synchronizer for LOAD/FETCH acknowledge signal is designed based on a 4-stage FSM. As shown in Fig. 4.18, the f2sACK signal high will last for PROCESS and WAIT states period in order to be captured by the rising edge of sCLK. A handshake signal, f2sACK$_{fb}$, is sent back to the control FSM of hash core to indicate a successful LOAD/FETCH. Within this approach we extended the standard hash interface [34] of each candidate to integrate this low-cost and simplified synchronizer, and the final reported layout area for each candidate will also include the overhead of this extended hash interface.

Table 4.4: The summary of design specifications of SHA-3 finalists

Algorithm	Implementation Descriptions
BLAKE-256	4 parallel **G** functions; 1-stage pipeline in permutation
Grøstl-256	Parallel **P** and **Q** with 128 GF-based AES SBoxes
JH-256	SBoxes **S0** and **S1** are implemented in LUT
Keccak-256	One clock cycle per round
Skein512-256	Unrolled 4 Threefish rounds

Power/Energy Approximation

In order to obtain accurate power profiles of each SHA-3 finalist, the ideal solution is to have separate power networks for each hash module. Since SASEBO-R test platform only supports a single power network for the chip, we use gated logics to force the inactive hash modules to enter into idle state by pulling down the clock and all the control and data signals. We estimated the static power dissipation of each module based on the area ratio and the standby power measured for the full chip. Note also that in 130nm, the static power dissipation is typically only a small portion of the complete power dissipation. In order to justify the power measurements, we have also compared them with the results from post-layout simulation and they closely match with each other as shown in Table 4.5.

Summary of Five SHA-3 Finalists Implementations

Table 4.4 summarizes the major implementation aspects of each SHA-3 finalist. The design decisions are made to achieve the primary optimization for *Throughput-to-Area ratio*. For details on the SHA-3 candidates please refer to the related specification documents on the NIST SHA-3 web site [113].

4.4.3 Silicon Implementation of SHA-3 ASIC

This section compares the synthesis and layout constraints of the SHA-3 ASIC.

The timing constraints are selected to optimize *Throughput-to-Area ratio*, using the methodology described in [61]. Although all the RTL designs are optimized for *Throughput-to-Area ratio*, depending on the different scenarios we may put different constraints during the synthesis and layout which may greatly affect the quality of the ASIC results. For synthesis, we evaluate four design points for every implementation (as described in Section 4.3.1).

As shown in the Fig. 4.19(a), all the dash lines connect to the points with highest *Throughput-to-Area ratio*, which are always the *MaxSpeed* point except for Grøstl whose optimal point is *Tradeoff1*.

After place&route, the routing delay will lower down the maximum frequency, and together with the added routing area the *Throughput-to-Area ratio* will be reduced. As shown in the equation below, the *weight* as the degradation factor will be always larger than one for post-layout.

$$[f_{max}/area]_{layout} = \frac{[f_{max}/area]_{synthesis}}{weight} \tag{4.2}$$

Ideally, we may relax the timing constraints of each hash module and obtain a uniform *weight* for all the candidates before and after layout. However, in practice by relaxing the timing constraint the degree of area decrease is not uniform for different candidates. This can also be observed in synthesis results in Fig. 4.19(a). Therefore we categorized the five SHA-3 candidates plus the SHA256 based on their achievable frequencies after layout into three groups: 250 *MHz* (JH and Keccak), 200 *MHz* (Grøstl and SHA256), and 125 *MHz* (BLAKE and Skein). As shown in Fig. 4.19(b), comparing the individual synthesis results with the final layout results of all candidates on the same die, we put larger *weight* to the high speed designs, JH and Keccak, with an average *weight* of 1.9 for all the designs. For

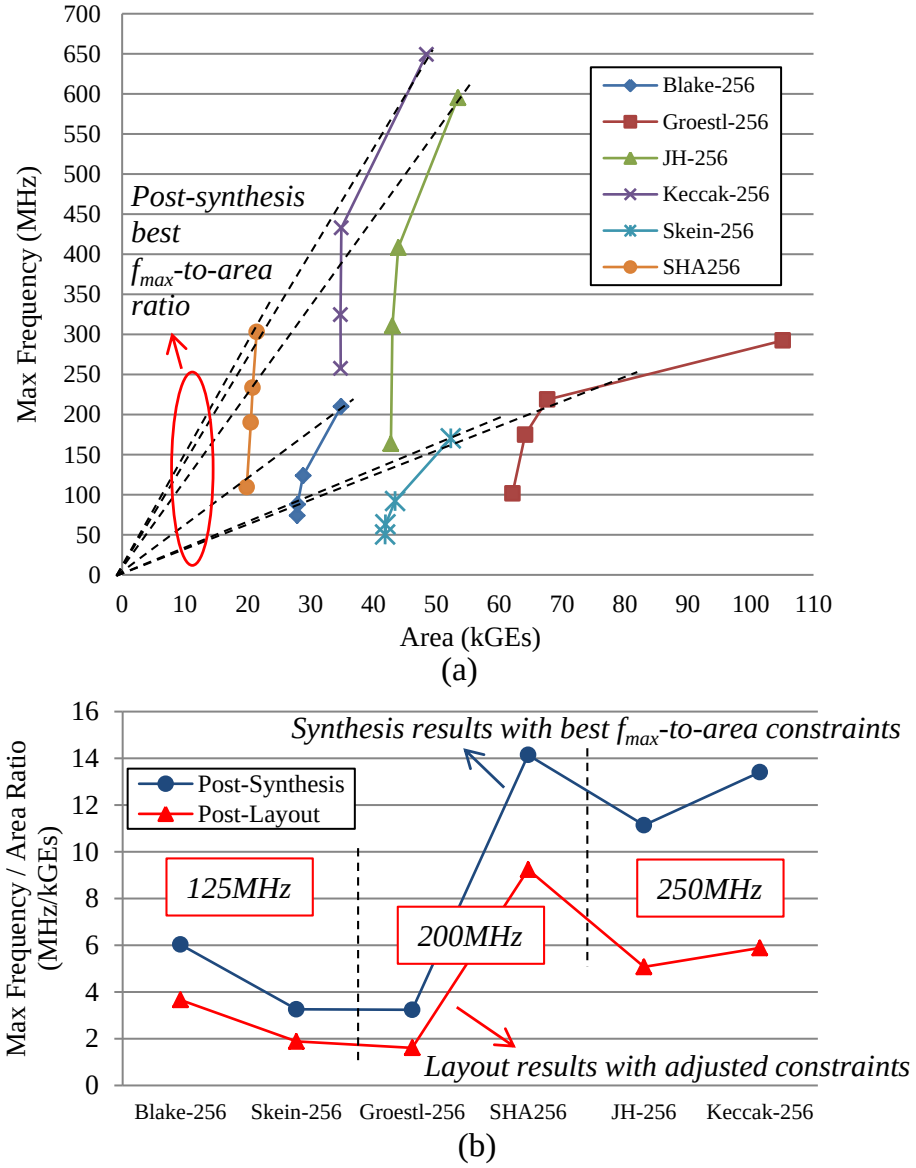


Figure 4.19: (a) Synthesis exploration of each SHA-3 candidate; (b) Weight factors to determine the final ASIC layout constraints.

high frequency designs after synthesis, more weight has been added for layout constraints to avoid explosion of the overall chip area.

The resulting layout, shown in Fig. 4.20, uses the IBM MOSIS 130nm CMR8SF-RVT standard cell library. The chip core area is 1.656 mm × 1.656 mm and overall chip die size is 5 mm^2 including pad cells. Out of seven metal layers, five metal layers are used for signal

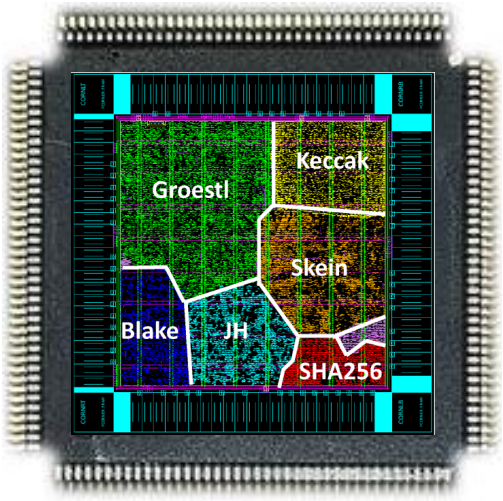


Figure 4.20: The SHA-3 ASIC layout and 160-pin QFP package.

and clock routings and the top two layers are used for power and ground. The overall chip core area utilization ratio after layout is 73%. The chip is packaged with 160-pin QFP to be compatible with the SASEBO-R board.

4.4.4 Analysis of ASIC Measurement Results

To understand the functional limits of each candidate at different operating voltages, we present the characteristic frequency *vs.* supply voltage shmoo plot in Fig. 4.21. Core supply voltage VDD is varied from 0.8V to 1.4V.

At nominal voltage, the measured frequency for all candidates meets targeted frequency. At higher supply voltage of 1.4V, maximum frequency measured from 25-stage S-RO is 250MHz. We were unable to verify the functionality beyond 350 MHz with available range of on-chip frequencies. Hence, at an operating voltage of 1.4V there exists possibility that JH and Keccak can run faster than 350Mhz. From the shmoo plot, we can rank all candidates in terms of frequency of operation. JH and Keccak can run at maximum speed of 350 MHz, followed by Grostl, BLAKE and Skein.

From the ASIC measurement results shown in Table 4.5, all the five SHA-3 candidates

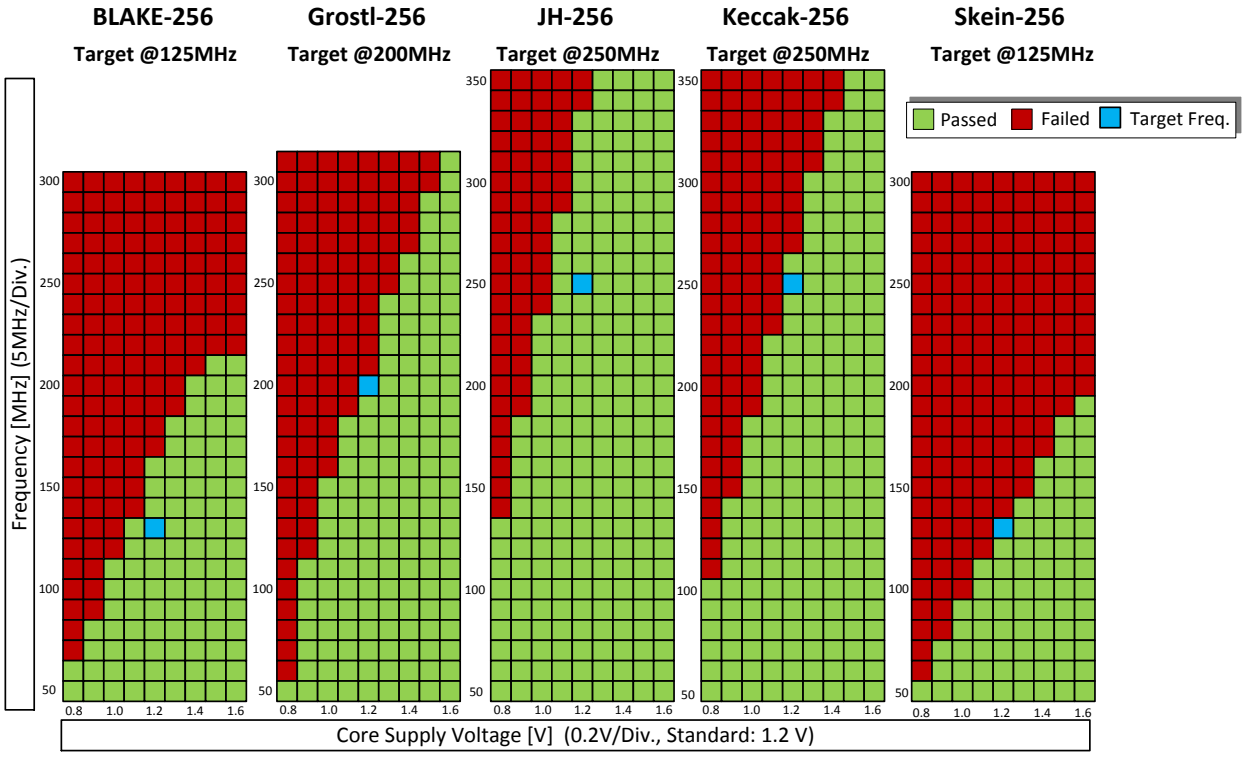


Figure 4.21: The Shmoo plot for frequency-voltage for the five SHA-3 finalists.

meet the target maximum frequency, and have a larger area but higher throughput than the reference SHA256; the maximum throughput of Grøstl and Keccak can almost reach 10 *Gbps*; Keccak is the best in hardware efficiency and energy efficiency; JH is the most power efficient SHA-3, closely followed by Keccak and BLAKE, but still less efficient than SHA256. The power consumption results are measured at fixed slow interface clock frequency of $1.5MHz$ and fast hash core clock at $50MHz$. Both of the clocks are provided by the control FPGA through on-board wire connections to the SHA-3 ASIC, so relatively slow interface clock frequencies are chosen for stability.

The latency and energy efficiency of SHA-3 finalists should be evaluated with different message lengths due to the different overheads in both of the hash initialization and finalization steps. We examined all the SHA-3 finalists with very short message and message lengths around the most common used Internet packet sizes (i.e. 576 and 1500 bytes) [31]. We compared the SHA-3 finalists without considering the interface overhead, which can

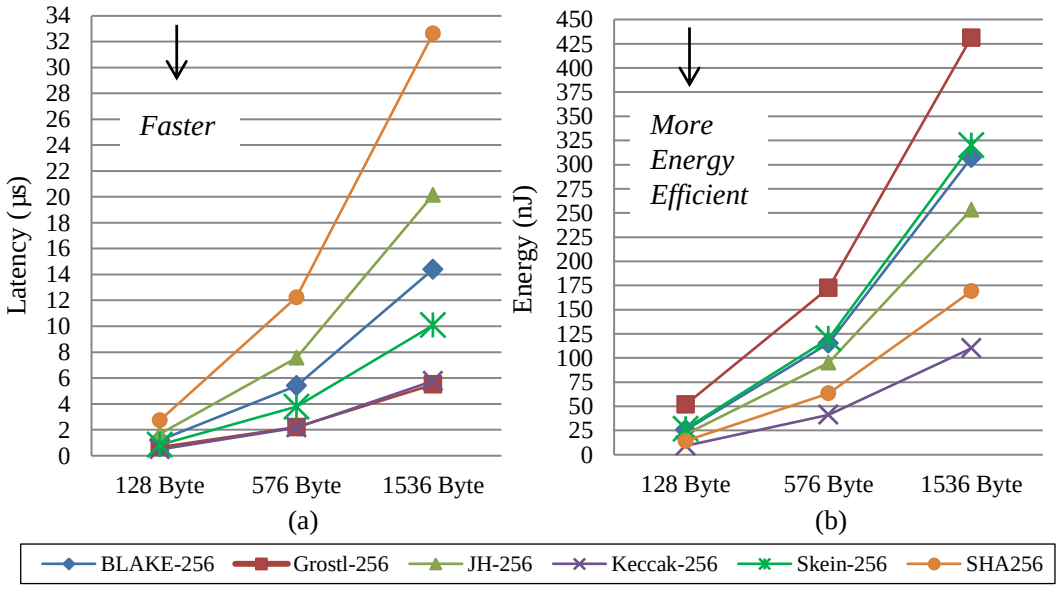


Figure 4.22: (a) The latency curve and for different packet sizes assuming ideal interface; (b) The energy curve for different packet sizes assuming ideal interface (the energy numbers are estimated based on average power measurements at slow and fast clock of 1.5 *MHz* and 50 *MHz*, respectively).

better examine the characteristics of SHA-3 candidates themselves. As shown in Fig. 4.22, Keccak and Grøstl are the fastest for all the three cases; Keccak is also the most energy efficient finalist. Fig. 4.22 also shows that the rankings of candidates almost do not change based on message length, although their differences grow at longer message lengths. The overhead of finalization step in Grøstl makes it slightly slower than Keccak when hashing short messages, but for the case of hashing long messages Grøstl becomes faster than Keccak. Note that we have listed the power/energy results for both of the ASIC measurements and post-layout simulation results for comparisons. Grøstl and Skein show relatively larger differences in these two cases; however, there might be sources of inaccuracy in both power models and measurement tools which are difficult to justify. We consider these power variations are less important issues since the order of power/energy efficiency for different SHA-3 finalists does not change as shown in Table 4.5.

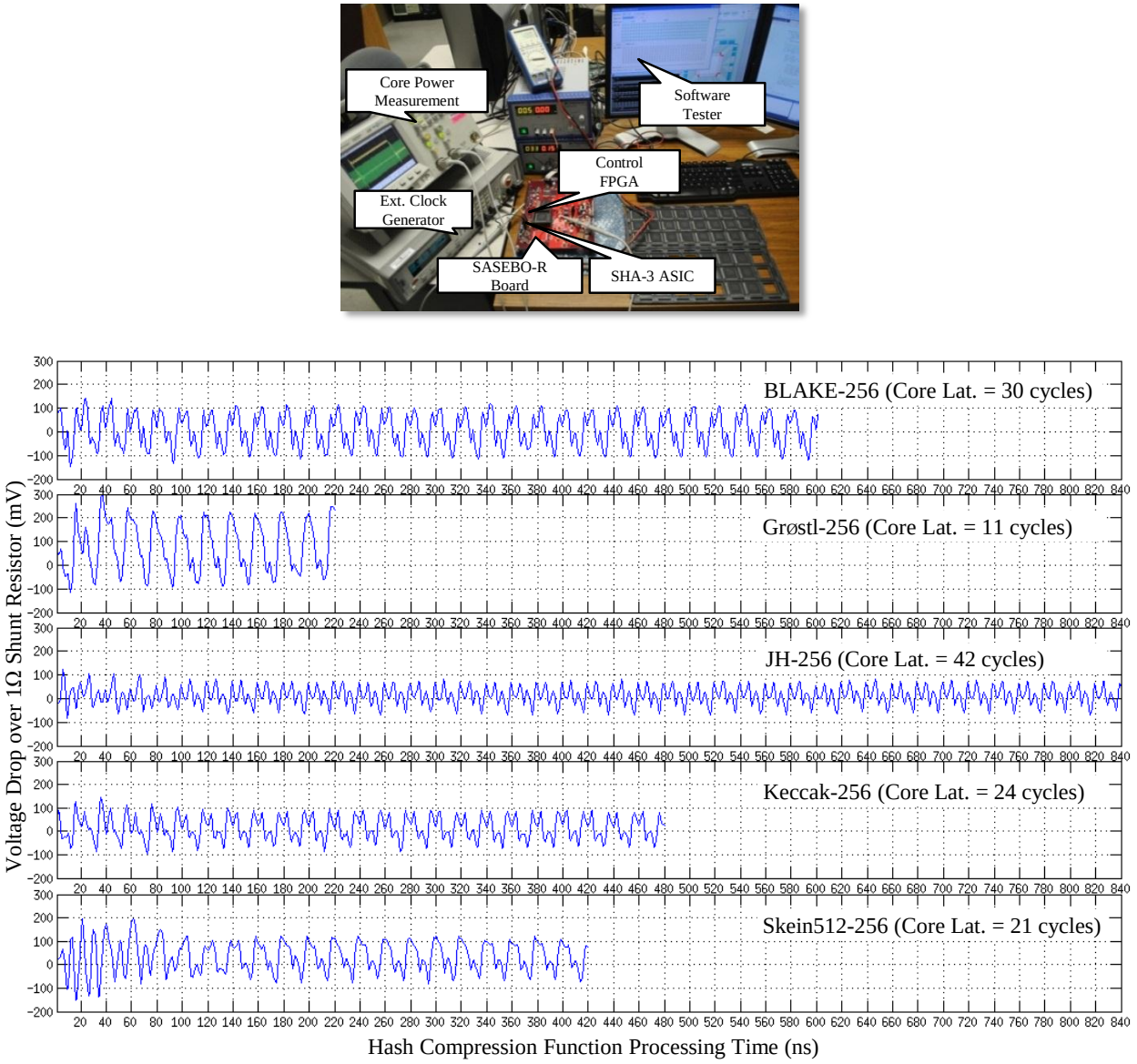


Figure 4.23: The SHA-3 ASIC testing environment and sample voltage drop (power) traces measured at slow and fast clock of 1.5 *MHz* and 50 *MHz*.

Table 4.5: ASIC Characterization of the SHA-3 ASIC chip in IBM MOSIS 130nm CMOS Technology with CMR8SF-RVT standard cell library

	Block Size	Core Lat.	Area[a]	Max Freq.	Tp	Tp/Area	Power[b]	Energy[b]	Power[c]	Energy[c]
	[$bits$]	[$cycles$]	[$kGEs$]	[MHz]	[$Gbps$]	[$kbps/GE$]	[mW]	[$mJ/Gbits$]	[mW]	[$mJ/Gbits$]
BLAKE-256	512	30	34.15	125	2.13	62.47	21.33	25.00	19.77	23.17
Grøstl-256	512	11	124.34	200	9.31	74.87	78.42	33.70	139.29	59.85
JH-256	512	42	49.29	250	3.05	61.83	12.57	20.63	13.01	21.35
Keccak-256	1024	24	42.49	250	10.67	251.05	19.12	8.96	19.78	9.27
Skein512-256	512	21	66.36	125	3.05	45.93	31.74	26.04	51.09	41.91
SHA256	512	68	21.67	200	1.51	69.54	5.18	13.76	5.05	13.42

a: the Gate Equivalent count is calculated by dividing the post-layout die area by the area of a NAND2XLTF (5.76 μm^2).

b: numbers are based on chip measurements of SHA-3 ASIC with slow chip interface clock at 1.5MHz and fast hash core clock at 50MHz.

c: numbers are based on post-layout simulation of SHA-3 ASIC with slow chip interface clock at 1.5MHz and fast hash core clock at 50MHz.

Note:

1: All five SHA-3 candidates are implemented with NIST SHA-3 Round 3 Specifications by January, 2011.

2: Each design's static power is estimated by multiplying the whole chip static power, 1.92 mW, with the area ratio of each design.

4.4.5 Summary

This section reported the *first* ASIC measurement results for hardware evaluation of SHA-3 finalists based on the Round 3 specifications published online in January, 2011. Moreover, as our SHA-3 ASIC has been designed to be compatible with SASEBO-R board, which is an open platform for side-channel attack analysis and widely distributed among the cryptographic hardware community. We have already distributed SHA-3 chips to other research groups and establish a public side-channel evaluation process on the SHA-3 ASIC implementations. As for power measurement setup in power analysis attacks, sample voltage drop (proportional to the power consumption) traces identifying the interested hash core operating period are shown in Fig. 4.23. The software tester running on a host PC, the control FPGA hardware on SASEBO-R, and the SHA-3 ASIC are all publicly available [64].

4.5 Conclusion

In this chapter we propose a methodology for SHA-3 ASIC evaluation and summarize our efforts in doing benchmarking in ASIC of five SHA-3 finalists. We present their rankings based on actual measurement performed on chip based on performance and power consumption. We analyze their trends under different hardware platforms. From our knowledge this is the first ASIC implementation of SHA-3 finalists.

Chapter 5

The Impact of Technology in Fair Comparison of Cryptographic Hardware

The growing emphasis on engineering aspects of cryptographic algorithms can be observed through recent advances in hash designs, which are strongly implementation-oriented. Fig. 5.1 demonstrates three important design fields in hash design: algorithm-specification, hardware architecture, and silicon implementation. Crypto-engineers are familiar with the relationship between algorithm- and architecture-level. However, the silicon implementation remains a significant challenge for the crypto-engineer, and the true impact of design decisions often remains unknown until the design is implemented.

In this chapter, we extend the SHA-3 hardware evaluation work by discussing two technology dependent issues which may significantly affect cryptographic hardware evaluation results. Section 5.1 discusses how to compare FPGA and ASIC results with fixed HDL designs and provides some insights in how to look at the SHA-3 FPGA and ASIC benchmarking results together. Section 5.2 discusses lightweight hash implementations and shows a more interactive process in designing and optimizing hash functions.

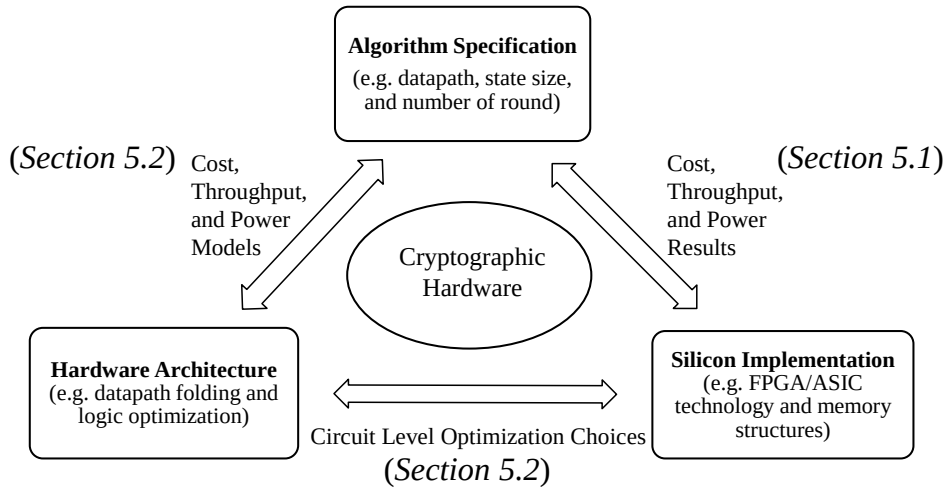


Figure 5.1: The interactive process in the development of lightweight cryptography.

5.1 Comparison of SHA-3 FPGA and ASIC Results

Both FPGAs and ASICs are widely used as targets for comparing SHA-3 hardware benchmarking process. However, the impact of target technology in SHA-3 hardware benchmark rankings has hardly been considered. A cross-platform comparison between the FPGA and ASIC results of the 14 second round SHA-3 designs demonstrates the gap between two sets of benchmarking results.

5.1.1 Introduction

Two major classes of hardware devices, Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs), were extensively studied during Round 2 SHA-3 hardware evaluation [14, 15, 43, 51, 52, 60, 74, 89, 105, 129, 130]. It is widely accepted that FPGAs and ASICs implementing the same design show different characteristics [99]. A hardware benchmarking process, therefore, starts by fixing the target technology, either ASICs or FPGAs, and then report the results based on selected metrics that are appropriate for the target technology. Several SHA-3 hardware rankings have been obtained in this manner. In this chapter we intend to address the question if the choice of target technology

can affect the resulting ranking between FPGA and ASIC designs built *based on the same HDL source code*.

In general, compared to ASICs, FPGAs offer many advantages including reduced nonrecurring engineering and shorter time to market. These advantages come at the cost of an increase in silicon area, a decrease in performance, and an increase in power consumption when designs are implemented on FPGAs. These inefficiencies in FPGA-based implementations are widely known and accepted, although there have been few attempts to quantify them. One exception is Kuon, who describes the gap between ASIC and FPGA in terms of area, performance, and power consumption [99]. Kuon compares a 90-nm CMOS FPGA and 90-nm CMOS standard-cell ASIC in terms of logic density, circuit speed, and power consumption for core logic. He finds that, for a representative set of benchmarks, the area gap between FPGA and ASIC is 35 times. He points out that the area gap may decrease when "hard" blocks in the FPGA fabric (multipliers, memories, and so on) would be used. The ratio of critical-path delay, from FPGA to ASIC, is roughly three to four times. The dynamic power consumption ratio is approximately 14 times and, with hard blocks, this gap generally becomes smaller.

In this chapter we report on a methodology to provide a consistent comparison between SHA-3 FPGA and ASIC designs with three major steps. First, we select the technology node for both FPGAs and ASICS as the starting point for our cross-platform evaluation. Second, we propose several metrics to approach a comparison between FPGA and ASIC results. Third, present an analysis of such results for 14 candidates implemented in ASIC and FPGA.

5.1.2 Related Work

The hardware evaluation of SHA-3 candidates has started shortly after the specifications and reference software implementations of 51 algorithms submitted to the contest became available. The majority of initial comparisons were limited to less than five candidates

[14, 43]. More comprehensive efforts became feasible only after NIST's announcement of 14 candidates qualified to the second round of the competition in July 2009. Since then, in both FPGA and ASIC categories, several comprehensive studies have been reported [15, 51, 52, 60, 74, 89, 105, 129, 130]. Matsuo *et al.* [89, 105] focused on the use of FPGA-based SASEBO-GII board from AIST, Japan. All the results are based on the prototyping results and real measurements on a Xilinx Virtex-5 FPGA on board. Gaj *et al.* [51, 52] conducted a much more comprehensive FPGA evaluation based ATHENA, which can generate multiple sets of results for several representative FPGA families from two major vendors. Baldwin *et al.* compared hardware implementations of different message digest sizes, including hardware padding, on a Xilinx Virtex-5 FPGA. Guo *et al.* [60] used a consistent and systematic approach to move the SHA-3 hardware benchmark process from the FPGA prototyping by [89, 105] to ASIC implementations based 130nm CMOS standard cell technology. Tillich *et al.* [129] presented the first ASIC post-synthesis results using 180nm CMOS standard cell technology with high throughput as the optimization goal and further provided post-layout results [130]. Henzen *et al.* [74] implemented several architectures in a 90nm CMOS standard cell technology, targeting high- and moderate-speed constraints separately, and presented a complete benchmark of post-layout results.

Table 5.1.2 compares these benchmarking efforts, and demonstrates that a comparison between FPGA and ASIC is hard because of several reasons. First, most groups do not share the same source codes. Second, the ASIC benchmarks do not use a common hardware interface. Third, the reported metrics do not allow a cross-platform (ASIC-FPGA) comparison. Although the joint work done by Matsuo *et al.* [89, 105] and Guo *et al.* [60] satisfy the first two conditions, still we believe that the chosen metrics are not well-suited for a cross-platform comparison between FPGA and ASIC benchmarks. All of the above issues motivate our work, namely an investigation of the (dis)similarity between FPGA and ASIC benchmarks for SHA-3 hardware candidates with 256-bit digest.

Table 5.1: Comparison of the related SHA-3 hardware benchmarking work in both FPGAs and ASICs

		FPGA	
	Matsuo [89, 105]	Gaj [51, 52]	Baldwin [15]
Own Source Code?	Yes	Yes	Yes
Technology Choices	Xilinx 65nm Virtex-5	Multiple FPGAs Xilinx & Altera	Xilinx 65nm Virtex-5
Hardware Interface	Defined standard 'handshake' interface	Defined standard 'FIFO'interface	Defined standard interface w/ HW padding
Chosen Metrics	Area, Throughput, Power, Energy	Area, Throughput, Throughput-to-area ratio	Area, Throughput, Throughput-to-area ratio
Design Flow	FPGA prototyping with measurements	Post-place & route simulation	Post-place & route simulation
		ASIC	
	Guo [60]	Tillich [129, 130]	Henzen [74]
Own Source Code?	Same as [89, 105]	Yes	Yes
Technology Choices	130nm CMOS Standard Cell	180nm CMOS Standard Cell	90nm CMOS Standard Cell
Hardware Interface	Same as [89, 105]	Assume infinite bandwidth interface	Assume infinite bandwidth interface
Chosen Metrics	Same as [89, 105]	Area, Throughput,	Area, Throughput, Energy
Design Flow	Post-layout simulation	Post-layout/synthesis simulation	Post-layout simulation

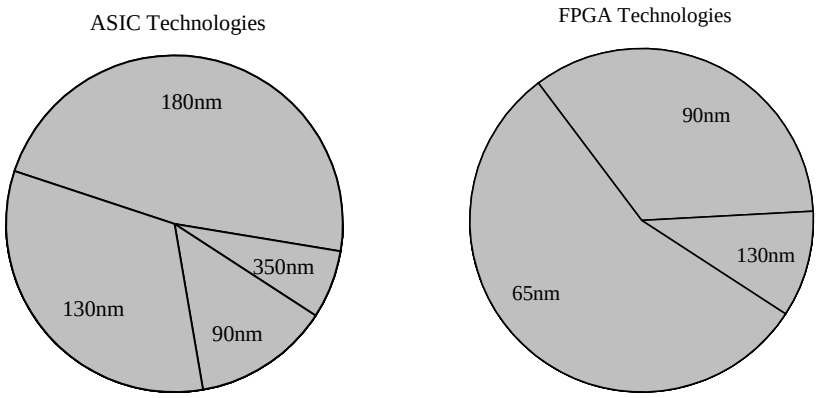


Figure 5.2: Technology nodes used for ASIC and FPGA hash implementations in the last 5 years.

5.1.3 Technology Node Selection for FPGAs and ASICs

It's not the intention of this work to pitch ASIC against FPGA. Instead, we want to evaluate how the performance numbers found on these two different technologies would be different assuming that someoone starts from the same RTL source code. This consideration affects how the target technologies for comparison are selected.

We have done a survey of hash hardware implementation papers published in CHES proceedings, Cryptology ePrint Archive and SHA-3 zoo in the past five years from 2005 (shown in Figure. 5.2). For around 90 reported hash implementations in FPGAs, around 56% of them are using 65nm FPGAs and 34% with 90nm FPGAs. For 61 ASIC implementations, 48% designs choose 180nm and 33% for 130nm. Thus, the most popular ASIC technology is several generations behind FPGAs, from 180nm to 65nm. Excluding high-end hardware components such as microprocessors, similar trends exist when looking at industry designed hardware. In our comparisons, we opted for the 65nm technology node for FPGA and the 130nm technology mode for ASIC.

We also evaluated the impact of technology scaling on FPGA and ASIC, i.e. we estimated the impact of more advanced technology nodes on our results. For FPGAs, the scaling factors are generally hard to quantify because different FPGA families may have drastically

different architectures. In [52], researchers have already demonstrated the influence of different technology nodes on the FPGA results for SHA-3 Round 2 candidates. For example, when moving from a 90nm Xilinx Spartan3E to a 65nm Xilinx Virtex-5, the basic logic element changes from 4-LUT to 6-LUT. In addition, the presence of hardened IP blocks, such as embedded memory (Block RAM), clocking management blocks and DSP functions, can lead to differences between two FPGAs within even the same technology node. Therefore, our comparisons of the 14 SHA3 designs in FPGA are specifically made for a Xilinx 65nm Virtex-5 FPGA. For other FPGA technologies, we recommend the use of an automated framework such as ATHENA [52, 53].

For ASICs, an almost linear scaling factor can be expected. In [60], we used CubeHash, one of the SHA-3 candidates, as a case study to evaluate the impact of different technology nodes (90nm vs. 130nm standard cell ASICs), different ASIC synthesis constraints and compare the post-synthesis results with post-layout results.

5.1.4 Comparison of FPGA and ASIC CAD Flows

In the FPGA CAD flow, all the 14 SHA-3 designs were implemented on Xilinx Virtex-5 (XC5VLX330-2FF1760) using the Xilinx ISE 12.2 software for all stages of the CAD flow. The synthesis was performed using ISE XST with default settings to perform speed optimization with normal effort. We changed the HDL options by disabling the tool to infer DSP blocks (which contain multiplier-accumulator circuits) and Block RAMs automatically from the RTL. These heterogeneous resources are specific to the Virtex device, and they complicate the analysis. Therefore, we restricted the synthesis tool from using these complex hard macro's. Placement and routing was performed using the standard effort level, and no timing constraints were placed on the design. After generating the post-place & route simulation model, we verified the functionality of each design and collect stimuli traces for power estimation with Xilinx XPower.

While the FPGA CAD flow is straightforward, the CAD flow for ASIC standard-cell imple-

mentations is significantly more complicated with more flexibility. We used the Synopsys Design Compiler (C-2009.06-SP3) to map the RTL codes to 130nm (FSC0G_D_SC_TP_2006Q1v2.0) technology. We use the typical case condition characterization of the standard cell libraries.

Although all the RTL designs are optimized for high throughput, depending on the different application scenarios we may put different constraints during the synthesis and layout which may then greatly affect the quality of the ASIC results. We evaluate four design points for every implementation (as described in Section 4.3.1).

The Synopsys IC Compiler (C-2009.06-SP5) is used for the back-end process. For all the designs we start with 85% utilization of the core area. The *utilization* is the ration of the active chip area (gates) to the total chip area (gates, wires, and empty space). The 130nm technology uses 8 metal layers. In general, more metal layers allow for a denser interconnect, and hence a more optimal use of die area. Overall, we reused the recommended scripted flow from Synopsys Reference Methodology [128]. The area and timing results are obtained from post-layout steps. Power results are obtained from Prime Time (C-2009.06-SP3) after passing post-layout simulation.

After implementing each design in the ASIC and FPGA flow, the area, delay, and power of each implementation were compared. For ASIC area, we only consider the final core area of the layout without I/O pad cells in Gate Equivalent (GE); the FPGA area is directly retrieved from the post-place & route report in Slices. The critical path delay of both FPGA and ASIC are derived from static-timing analysis assuming worst case operating conditions.

The power metric for FPGA and ASIC includes the static and dynamic portions of the estimated power consumption. We made the following adjustment to make the metric comparable between ASIC and FPGA. The static power of the FPGA is scaled by the fraction of the core FPGA area used by the design. With this, we attempt to compensate for the portion of the FPGA that is not used by a design. Furthermore, a 65nm FPGA technology will have a significantly higher leakage than a 130nm ASIC technology.

We note once more that it's not our intention to pitch ASIC against FPGA, but instead

Table 5.2: Proposed metrics for SHA-3 hardware benchmarking

	Description	Note
Metric 1	Maximum Throughput	Useful for both customized & fixed IP cases; Show the performance limits of designs by stretching technology.
Metric 2	Achievable Throughput per Area	Useful for both customized & fixed IP cases; Proportional to (f_{max} /area) which shows the price to pay for stretching technology.
Metric 3	Power and Area under Fixed Throughput	Useful for only fixed IP case; Compare designs considering technology influences but without stretching technology.

of investigating how the selection of either ASIC or else FPGA may affect the ranking of SHA-3 candidates.

5.1.5 Comparison of FPGA and ASIC Rankings

In this section, we will discuss how to select meaningful metrics to produce comparative results for both FPGAs and ASICs.

To conduct a meaningful comparison, we believe an application scenario must be chosen. Two cases can be considered. The first one is the "customized IP" case, which means the designer will use application-specific information to constrain the FPGA and ASIC CAD flow to achieve the best possible hardware area and performance results of a given IP in a given application. The second one is the "fixed IP" case. In this case, system designers will just reuse a 'pre-made' IP and adapt them to their requirements only by adjusting the clock frequency. In this work, we will consider the latter case. This leads to the three metrics summarized in Table 5.1.5.

For each chosen metric we provide the *relative ranking* of 14 Second-Round Candidates. Each column in the graph of ranking is normalized with respect to the lowest number of that

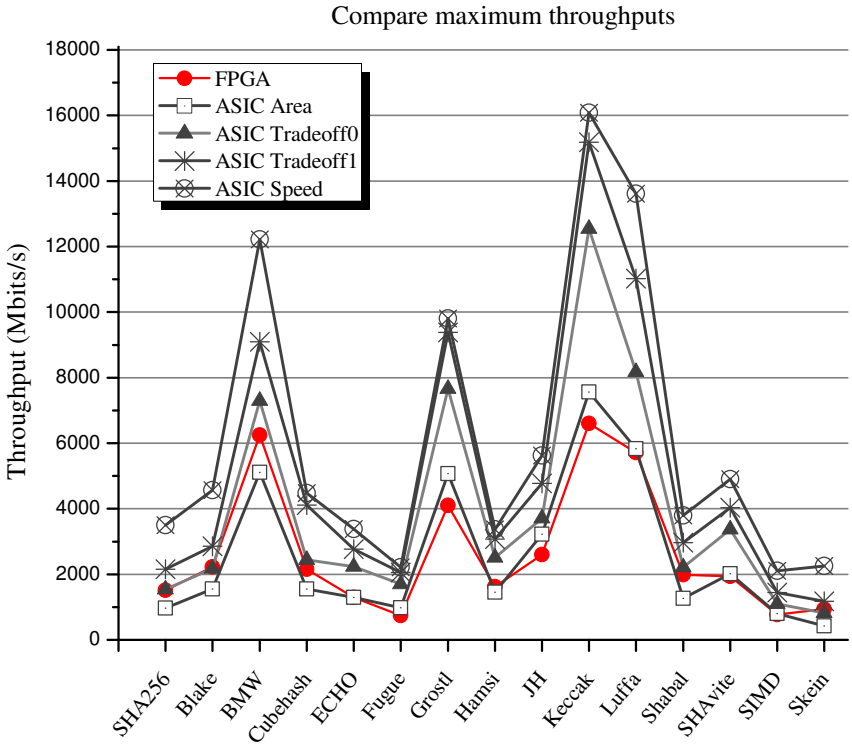


Figure 5.3: Comparison of the maximum throughput between ASICs and FPGAs

column. The model of rankings exhibits the relative distances among consecutively ranked candidates since some of the designs have very close results which can all be considered as equally good ones. In this way, we can categorize all the candidates into several small groups.

Metric 1: Maximum Throughput

The first metric compares the maximum throughput of different implementations when affected by different technologies and constraints. Since all the 14 Round 2 SHA-3 candidates are designed with high speed optimization in mind, this metric shows the potential of each candidate (see Figure 5.3).

From Figure 5.4, we can observe that the rankings of the algorithms under maximum throughput metric are quite uniform between FPGA and ASIC. Only small variations are found because of the impacts of different ASIC backend process constraints to different

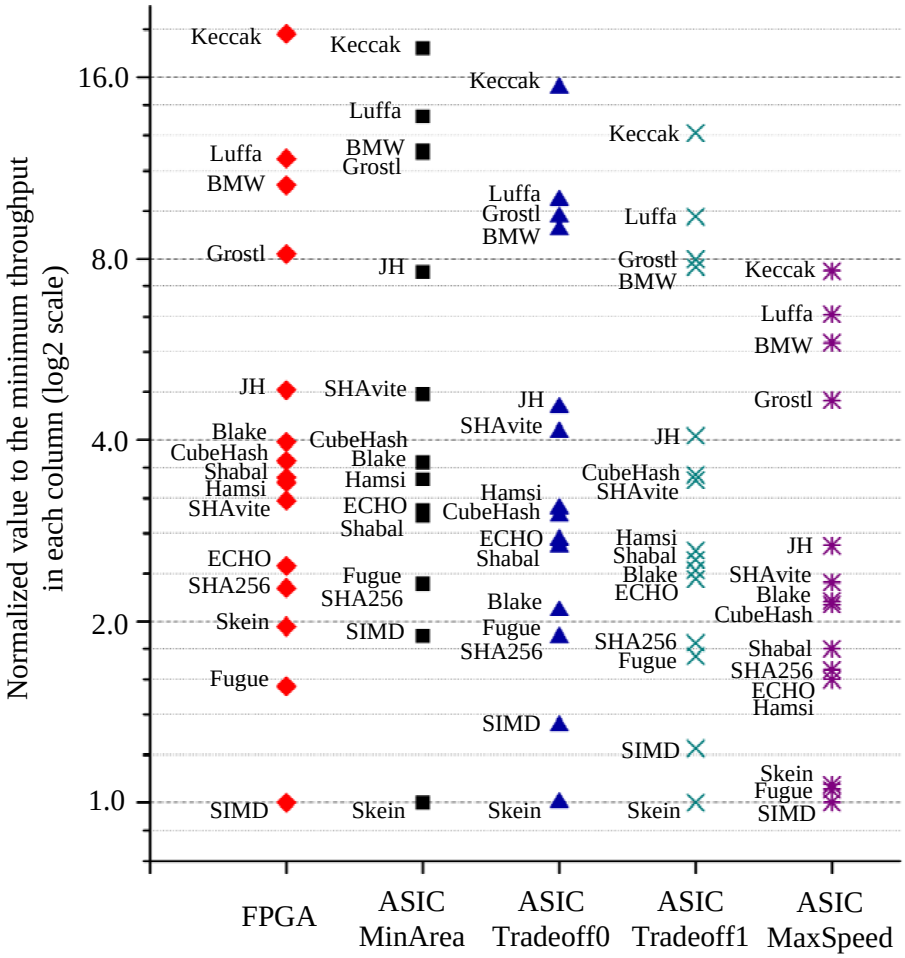


Figure 5.4: The ranking of relative maximum throughput in FPGAs and ASICs

algorithms with very similar area. For both FPGA and ASIC, Keccak is the best one in terms of maximum throughput, and there are four candidates, Keccak, Luffa, BMW, Grøstl, standing out. In Figure 5.4 we can also observe how the user's defined backend process constraints will affect the rankings once we fix the ASIC technology .

Metric 2: Achievable Throughput per Area

In metric 2, we compare the relative achievable throughput per area between ASICs and FPGAs.

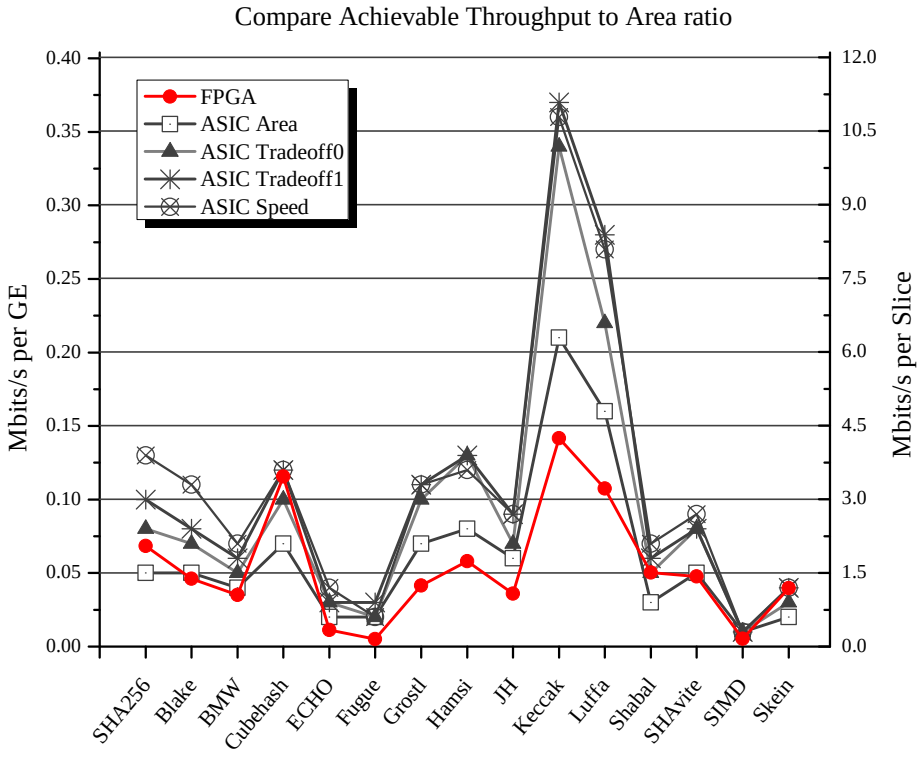


Figure 5.5: Comparison of the achievable throughput per area between ASICs and FPGAs

From Figure 5.5, it can be seen that for most of the 14 SHA-3 Round 2 candidates, ASIC Tradeoff1 case has the highest achievable throughput per unit of area and therefore provides an efficient trade-off point between area and throughput.

From Figure 5.6, we can observe that the rankings of the algorithms under achievable throughput per area metric have some differences between FPGA and ASIC. One of the major causes for these dissimilarities is the way to calculate the FPGA and ASIC area. Due to the fundamental architectural differences between FPGA and ASIC, it is inaccurate to transfer the basic element, *Slice*, for Xilinx FPGA as the area unit into GE counts in ASIC. Besides, the critical paths resulted from the existed interconnect networks inside the FPGA can be also an influential variant compared with those in customized ASIC layout. We think these two causes may roughly explain the big difference in rankings for CubeHash between FPGA and ASIC. A more detailed analysis to understand these dissimilarities is still important, and is part of our ongoing work.

This metric helps us to pick the most efficient ASIC implementation as the 'fixed IP' that we will use for point-to-point comparison between ASIC and FPGA. Recall that each SHA-3 design has four different ASIC implementations (MaxSpeed, MinArea, TradeOff0, TradeOff1), while there is only one single FPGA implementation. Therefore, the question becomes which ASIC implementation should be finally chosen to compare the FPGA and ASIC results. The four ASIC implementations include 2 boundary points, at minimum area and maximum speed. These are extreme cases that are usually avoided in practical design. Instead, we opt to use the so called '*sweet spots*' in the ASIC area-delay curve where there is an optimal trade-off between throughput and area. This is especially desirable in a 'fixed IP' scenario when the constraints of the final application are not known beforehand. Note that by choosing default settings of Xilinx ISE tools the FPGA results obtained can also be considered as a good trade-off between are and speed.

Metric 3: Power and Area under Fixed Throughput

By using the analysis results for metric 2, we can now do a point-to-point comparison between FPGAs and ASICs for all the SHA-3 designs.

The third metric is motivated by the application scenario we mentioned earlier. We assume that the system designers are now considering the system integration of two sets of SHA-3 hardware IPs implemented in ASICs and FPGAs, respectively. Since all those IPs have the same interface and since the system required throughput is fixed, the next step is to figure out whether the selected IP can satisfy a given area and power budget. Therefore we first fix the throughput of each design at 0.2 Gbps. Next, we compare the area and power of the candidates.

It can be observed from Figure 5.7 that the rankings of the algorithms are quite different between FPGA and ASIC, especially in terms of power. This means that characteristics of different candidates scale differently when moved from FPGA to ASIC. In order to study this more closely, we provide a point-to-point comparison between FPGA and ASIC imple-

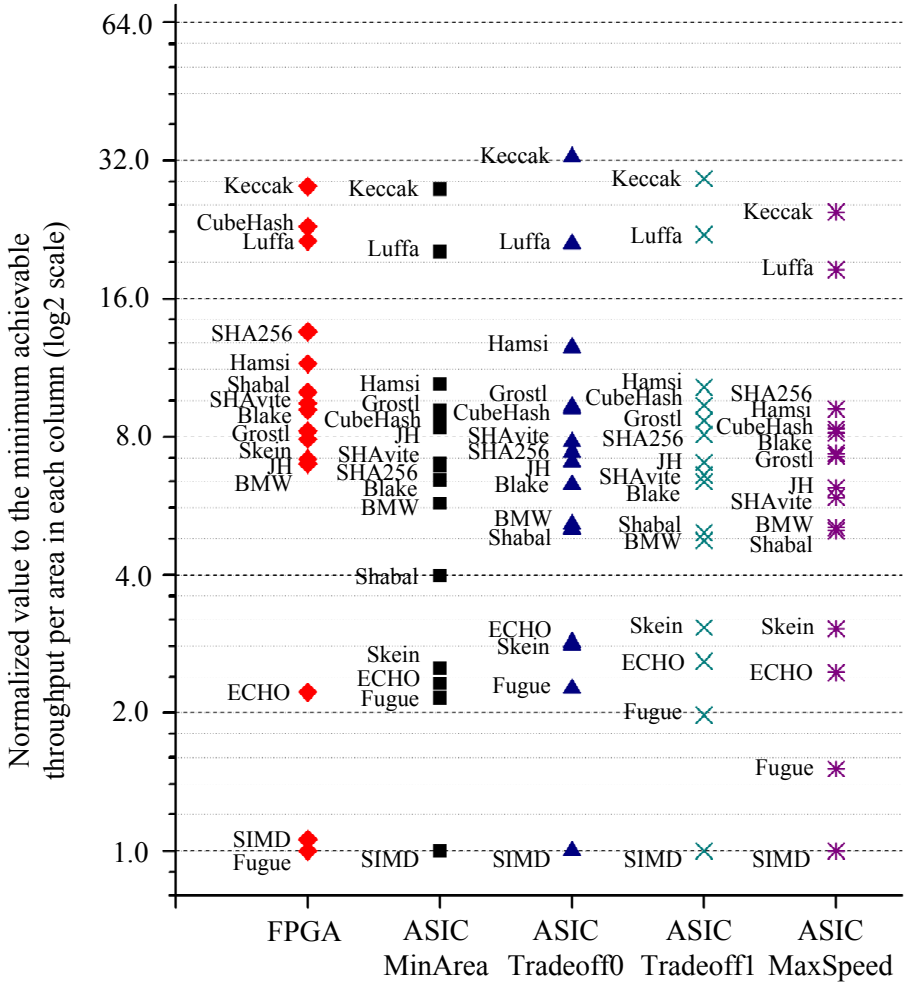


Figure 5.6: The ranking of relative achievable throughput per area in FPGAs and ASICs

mentation of each candidate. Figure 5.8 provides this comparison for area and achievable throughput, while Figure 5.9 shows the same for dynamic power and total power. Each of these metrics are discussed below.

Area. By default the unit of area in ASIC is Gate Equivalent (GE) and Slice for Xilinx FPGA. Internal data from FPGA vendors are needed if one tries to convert the slice to GE [99], but for simplicity here we only give estimation by using the ASIC-GE to FPGA-Slice ratio to denote the area gap between FPGAs and ASICs. The variation of this area ratio can be found in Figure 5.8, and range of this area ratio is from 16.46 to 55.37 with an

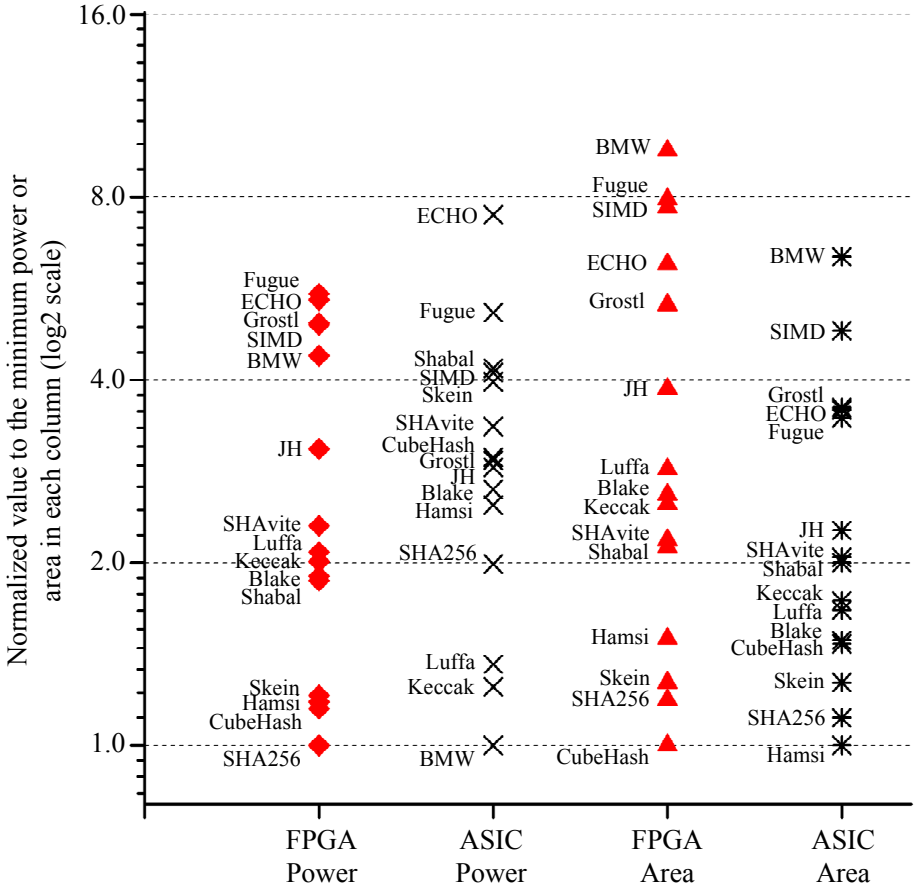


Figure 5.7: The ranking of relative power and area with fixed throughput at 0.2Gbps

average ratio of 28.64.

Throughput. The comparison results of achievable throughput ratio are shown in Figure 5.8. The ratio is ranged from 1.25 to 2.76, and the average is 1.92, which means when you shift from 65nm FPGA to 130nm ASIC, the maximum throughput in average will increase by 92%. This gap is much smaller compared to the previously reported numbers by Kuon [99] because they use the same technology node for ASIC and FPGA. In our case, FPGA is able to close the gap because is uses a more efficient technology.

Power. Even after scaling the FPGA static power consumption proportional to the FPGA area, we still find that the static power in FPGA can be as high as 92.1% for the BMW design with a minimum 37.2% for the Skein design. In ASICs, the static power contributes

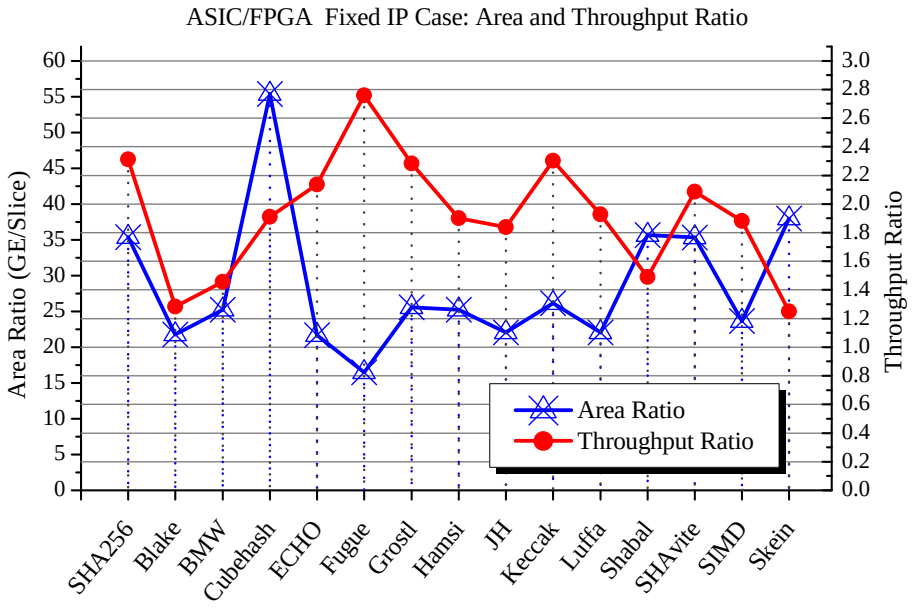


Figure 5.8: Comparison of the ASIC/FPGA area and achievable throughput ratio

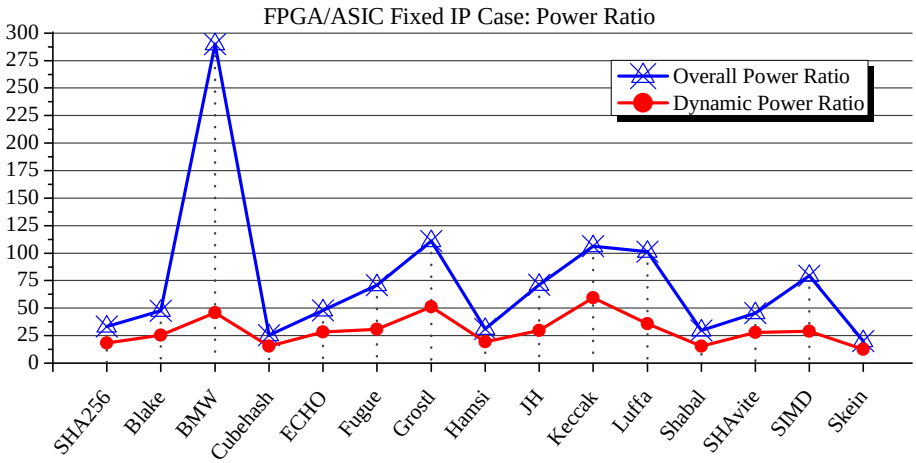


Figure 5.9: Comparison of the FPGA/ASIC overall power ratio with dynamic power ratio

less than 1% for all designs. The overall power ratio ranges from 20.10 to 289.99 with an average of 74.08, while the dynamic power ratio ranges from 12.64 to 59.34 with an average 29.59. So, even for dynamic power the FPGAs still consume 29.6 times of ASIC power in average from our SHA-3 benchmarking circuits.

5.1.6 Summary

In this section, we study the difference between FPGA and ASIC ranking for 14 SHA-3 Round 2 candidates. Three metrics are carefully selected to deliver meaningful comparison results for SHA-3 FPGA and ASIC implementations.

This work shows that ASIC and FPGA designers may come to different conclusions when it comes to making a statement on the most efficient SHA-3 candidate in hardware. However, each of ASIC and FPGA SHA-3 designs offer a similar design space (tradeoffs of around 7 times between most and least efficient ones in both area and power metrics as shown in Figure 5.7).

This work also lends some insights on how to look at SHA-3 hardware benchmarking results in different platforms. In cases where the platform is already fixed (ASICs or FPGAs), one should exclusively rely on FPGA-specific or ASIC-specific benchmarks, depending on the chosen platform. Conclusions on ASIC implementations based on FPGA results, or vice versa, will almost certainly be inaccurate. In some other cases, where you are looking to understand the SHA-3 candidates and where you do not yet have chosen a platform, it will be equally interesting to compare both the ASIC and FPGA SHA-3 results, because they point out different aspects of SHA-3 hardware implementations.

Table 5.3: FPGA and ASIC results with fixed throughput at 0.2 Gbps

	Block Size	Core Latency (cycles)	Work Freq. (MHz)	ASIC			FPGA		
				Area (GEs)	Max Freq. (MHz)	Power (mW)	Area (Slices)	Max Freq. (MHz)	Power (mW)
SHA256	512	68	26.6	26167	465.1	2.20	740	201.1	73.47
BLAKE	512	22	8.6	35062	122.7	2.93	1612	95.6	139.80
BMW	512	2	0.8	149858	35.5	1.11	5935	24.4	321.89
CubeHash	256	16	12.5	34443	257.0	3.31	622	134.6	84.36
ECHO	1536	99	13.0	83747	178.3	8.30	3864	83.5	398.54
Fugue	32	2	12.5	81343	128.5	5.73	4941	46.6	406.87
Grøstl	512	10	3.9	84607	183.2	3.28	3308	80.2	364.36
Hamsi	32	4	25.0	23484	384.6	2.77	930	202.3	86.53
JH	512	36	14.1	53055	335.6	3.18	2406	182.6	225.93
Keccak	1024	24	4.7	40712	355.9	1.39	1556	154.7	147.99
Luffa	256	9	7.0	39152	387.6	1.51	1774	201.0	152.85
Shabal	512	47	18.4	47051	272.5	4.64	1319	182.8	137.26
SHAvite	512	38	14.8	47887	299.4	3.72	1356	143.5	169.12
SIMD	512	46	18.0	113202	129.9	4.56	4790	69.0	362.12
Skein	256	21	16.4	29931	96.3	4.41	788	77.1	88.65

5.2 Cost Analysis of Lightweight Hash ASIC Designs

Cryptographic algorithm and protocols need to be tailored for implementation in many constrained environments, including RFID tags, wireless sensors, smart cards, and mobile devices. Lightweight Cryptography is a generic term that captures new efforts in this area, covering lightweight cryptography proposals as well as lightweight implementation techniques. This section demonstrates the influence of technology selection when comparing different lightweight hash designs and when using lightweight cryptography techniques to implement a hash design. First, we demonstrate the impact of technology selection to the cost analysis of existing lightweight hash designs through two case studies: the new lightweight proposal Quark [8] and a lightweight implementation of CubeHash [22]. Second, by observing the interaction of hash algorithm design, architecture design, and technology mapping, we propose a methodology for lightweight hash implementation and apply it to Cubehash optimizations. Finally, we introduce a cost model for analyzing the hardware cost of lightweight hash implementations.

5.2.1 Introduction

The tight cost and implementation constraints of high-volume products, including secure RFID tags and smart cards, require specialized cryptographic implementations.

Low-cost and power-efficient implementations of existing cryptographic standards (e.g. AES [47] for block ciphers and SHA [45, 115] for hash functions) have been extensively researched. However, the outcome of this research seems to imply that most of the established cryptographic standards cannot satisfy the requirements of extremely constrained applications, such as RFID tags [29]. In these applications, computational power, memory footprint, and power supply, all are limited.

Lightweight Cryptography is a recent trend that combines several cryptographic proposals that implement a careful tradeoff between resource-cost and security-level. For Lightweight

Hash implementations, lower security levels mean a reduced level of collision resistance as well as preimage and second-preimage resistance compared with the common SHA standard [112].

This chapter extends our previous work [63] in identifying the technology dependence of lightweight hash implementation cost by including more comprehensive cost analysis of lightweight hash designs. First, by mapping an existing lightweight hash proposal to different technology nodes and standard-cell libraries, we illustrate how important technology-oriented cost factors can be taken into account during analysis. Furthermore, through comparative study of the synthesized netlist we explain why different technology nodes have a larger influence than different standard-cell libraries at the same node. Second, by studying the interaction between hardware architecture and silicon implementation, we demonstrate the importance of low-level memory structures in lightweight hash design. As a case study, we show how to optimize CubeHash [22], a SHA-3 Round 2 candidate with a high security level, to fit in 6,000 GEs by combining bit-slicing (at hardware architecture level) and proper memory structures (at silicon implementation level). Fine-grained cost analysis of different components of bit-sliced CubeHash designs is performed, which provides a clearer view of the control overhead brought by the datapath folding. Third, we built several cost models for area, performance and energy metrics for lightweight hash designs and provide guidelines for both lightweight hash developers and hardware designers.

5.2.2 Overview of Lightweight Hash Implementations

Lightweight hash function designs fall in two research categories: lightweight hash proposals and lightweight implementation techniques.

Since DM/H/C-PRESENT [29] hash functions were proposed at CHES2008, we have seen several new lightweight hash proposals afterwards. DM/H/C-PRESENT [29] hash functions based on PRESENT block cipher and optimize for different hash constructions. Digest sizes from 64-bit to 192-bit can be achieved for a hardware cost of 2213 GEs (gate-equivalent) to 4600 GEs in 180 nm technology. ARMADILLO [12] hash family was first proposed at

CHES2010 as a dedicated hardware optimized hash proposal. It provides several variants providing digests between 80- and 256-bit with area between 2923 and 8653 GEs. The Quark [8] hash family is based on a sponge construction with a digest size from 128-bit to 224-bit and area from 1379 GEs to 4640 GEs in 180 *nm* technology. The most recently published SPONGENT [28] and PHOTON [58] hash families are also based on sponge construction, and for the first time both of them offer one variant under 1000 GEs. SPONGENT is based on a wide PRESENT-type permutation with a digest size from 88-bit to 256-bit with very small footprint in hardware from 738 to 1950 GEs, respectively. PHOTON has an AES-like internal permutation and can produce digest size from 64-bit to 256-bit with very close hardware cost as SPONGENT from 865 to 2177 GEs.

The ongoing SHA-3 competition aims at selecting the next generation of hash standard for general applications with high demands on security requirements. According to [65–67], the five SHA-3 finalists in the current phase and even the fourteen Second Round SHA-3 candidates [60, 62] are unlikely to be considered as lightweight hash candidates due to their high cost in hardware (more than 10,000 GEs) [8]. Nevertheless, we expect there will be additional effort dedicated to lightweight implementations of SHA-3 finalists. For some earlier work on existing hash standards, the smallest SHA-1 implementation with 160-bit digest costs 5,527 GEs in 130 *nm* technology [115]; SHA-256 with 256-bit digest can be implemented with 8,588 GEs in 250 *nm* technology [87].

To summarize previous works, two observations can be made. First, the new lightweight hash proposals emphasize the design of simplified hash core functions, rather than optimizing the implementation. Second, existing lightweight implementations focus on fine-grained or algorithm-specific optimizations. They do not provide general guidelines of how a given hash algorithm can benefit most from high level hardware architectural optimizations and low level technology optimizations.

In this work, we focus on the technology impacts to the cost analysis of lightweight hash designs and their relation to lightweight hash implementation techniques. Indeed, mapping

a standard hash algorithm into a lightweight implementation is at least as important as stripping down hash algorithms into lightweight-security versions.

Comparison between existing lightweight hash designs and the lightweight optimization of CubeHash described in this work can be found in the graph below:

5.2.3 Lightweight Hash Comparison Issues

Due to the nature of lightweight hash designs they are very sensitive to small variations in comparison metrics. For example, several of them claim area around 1,000 GEs with power consumption around 2 μW. However, it appears that most lightweight hash designers only discuss algorithm implementations at logic level, and they make abstraction of important technological factors. This makes a comparison between different lightweight hash designs difficult or even impossible.

To understand the issues, one should first identify whether the selected metrics are dependent or independent of the technology. Below we discuss several metrics that are commonly used to compare different lightweight hash proposals.

Area

Most of the lightweight hash papers compare the hardware cost by using the post-synthesis circuit area in terms of GEs. However, the circuit area estimation based on GEs is a coarse approach; it ignores the distinction between control, datapath, and storage, for example. Moreover, GEs are strongly technology dependent. In the following two case studies of Quark [8] and CubeHash [22], we will demonstrate different aspects of technology dependence of area cost.

- Standard-cell Library Impact. This is brought by different technology nodes and different standard-cell libraries. The ASIC library influence can be found by using the same

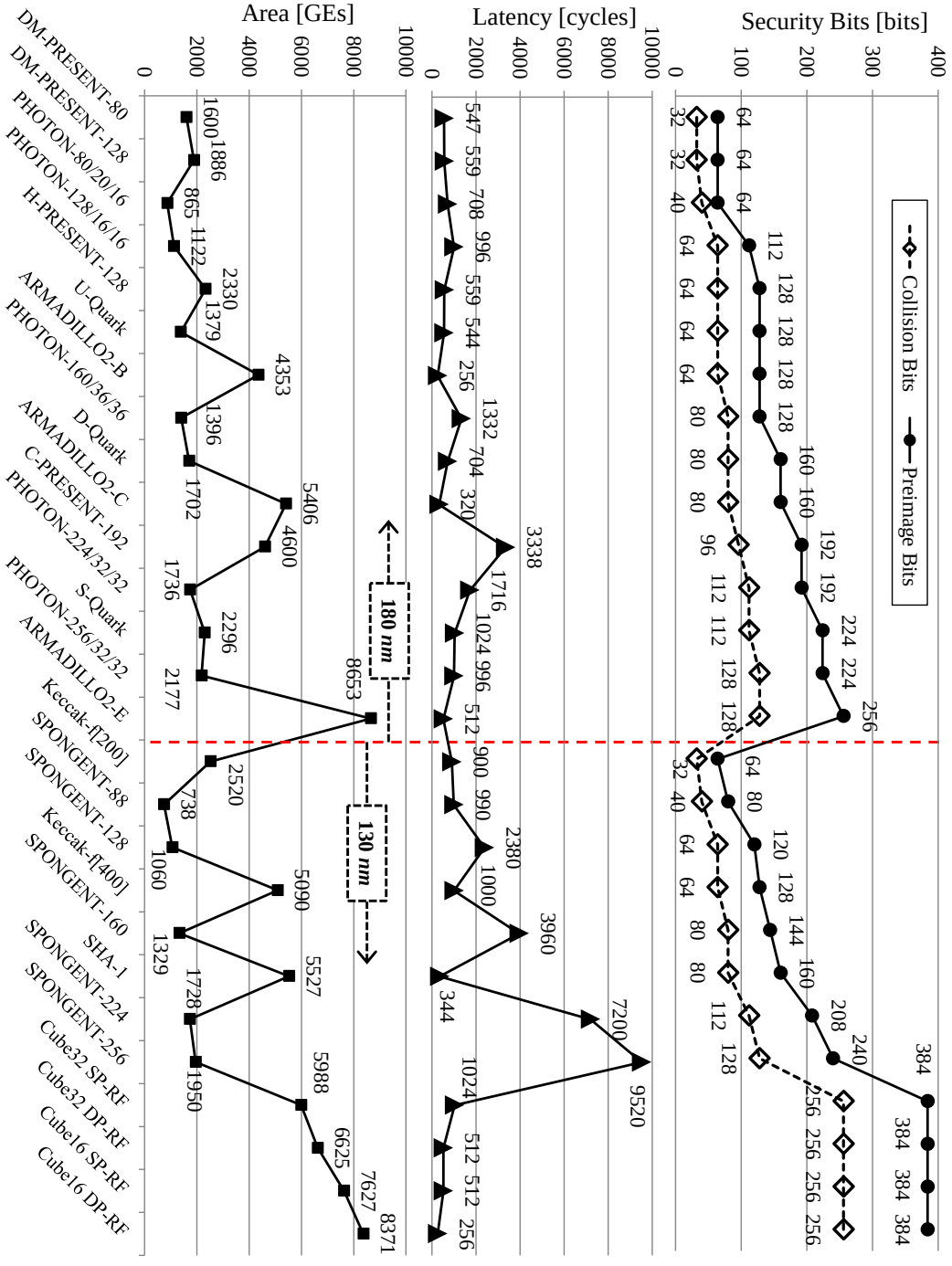


Figure 5.10: Summary of existing lightweight hash implementations (see Table 5.9 for more comparison metrics).

Table 5.4: Compare the characteristics of different ASIC technology nodes from some commercial standard-cell libraries [132].

Technology Node	Gate Density $[kGEs/mm^2]$	Power $[nW/MHz/GE]$
180 nm	125	15.00
130 nm	206	10.00
90 nm	403	7.00
65 nm	800	5.68

synthesis scripts for the same RTL designs in a comprehensive exploration with different technology nodes and standard-cell libraries. As found in the Quark case study in Section 3, the cost variation range caused by changing standard-cell libraries can be from -17.7% to 21.4%, for technology nodes from $90nm$ to $180nm$.

- Storage Structure Impact. Hash designs are state-intensive, and typically require a proportionally large amount of gates for storage. This makes the selection of the proper storage structure an important tuning knob for the implementation of a hash algorithm. For example, we implement several bit-sliced versions of CubeHash [19–21], and show that the use of a register file may imply an area reduction of 42.7% area reduction compared with a common flip-flop based memory design.

Power

The power consumption is another commonly used metric which is strongly correlated to the technology. For the standard-cell library impact, we can see from Table 5.4 the power efficiency in terms of $nW/MHz/GE$ differs by a factor of two to three times across different technology nodes. For the storage structure impact, as illustrated in our case study of CubeHash in Section 4, the power variation range can be from -31.4% to 14.5% at $130nm$ technology node. Therefore, it is important to provide proper context when comparing the power estimation results for different lightweight hash implementations.

However, in general we think that the power consumption is a less important metric in comparing different lightweight hash designs. Take the most popular and ultra-constrained RFID application as an example, which has the power budget for security portion as 27 μW at 100KHz [46,115]. Just by looking at the average power number in Table 5.4, we can easily get a rough estimation of the area needed to consume 27 μW would be 18k GEs, 27k GEs, 39k GEs and 48k GEs at 180nm, 130nm, 90nm and 65nm, respectively. Therefore, all of the lightweight hash proposals which are much smaller should satisfy the power requirement.

Energy

The energy consumption of a hash algorithm is an integral of the power consumption over the hashing period, which typically consists of three major steps: initialization, core hashing and finalization. Compared with the power metric, energy efficiency metric has more practical meaning since many lightweight applications (e.g. mobile handsets and wireless sensor nodes) are battery powered. We have noticed that several existing lightweight hash proposals have high timing overhead in the initialization/finalization procedures besides the core hashing operations. This means the energy efficiency of hashing short and long messages may have large differences. Similar as the power metric, energy metric is also strongly technology dependent.

Latency

The latency of hash operations is measured in clock cycles and is technology independent. However, one related metric, Throughput, may be technology dependent if the maximum throughput is required since the maximum frequency of a design is technology dependent. Since in most lightweight hash targeted applications, especially tag-based applications, hashing a large amount of data is unlikely to happen or the operating frequency is fixed at a very low value (e.g. 100 KHz for RFID), latency as a technology independent metric should be sufficient to characterize the lightweight hash performance. The latency metric is very

often simplified as cycle counts of the core hashing period for long messages; however, as some lightweight hash algorithms have non-negligible overhead for the initialization and finalization steps, the overall latency for short message hashing should be treated separately.

Summary

As discussed above, only the latency metric is independent of technology. Power is technology dependent metric but it is a much less important metric than area. Although energy metric is technology dependent, it is an important metric for practical applications. Therefore, the rest of the work will focus on the technology dependent analysis of the hardware cost and energy consumption of lightweight hash designs. The technology impacts of standard-cell libraries and storage structures are measured in a quantitative way through two concrete case studies of Quark and CubeHash designs.

5.2.4 Lightweight Hash Implementation Methodology

Below we propose a methodology for lightweight hash designs and created a cost model to help designers understand when changing memory structure becomes necessary in order to further reduce the circuit area.

Hardware Architecture Optimization: Datapath Folding The complexity of hash algorithms determines the ALUs used in the datapath to complete complex arithmetic operations. For some hash algorithms with 32-bit datapath, these ALUs may become the dominant factor in the overall cost.

Fig. 5.11 shows an example hash datapath with four logic operations between the IN and OUT states. There are two common ways to reduce the area of ALUs in this datapath. First, through horizontal folding, we can save half of the ALUs at the cost of introducing control overhead for selecting the inputs for each round of operations. Second, through vertical folding (also known as bit-slicing), we can cut the bit width of processing elements and this

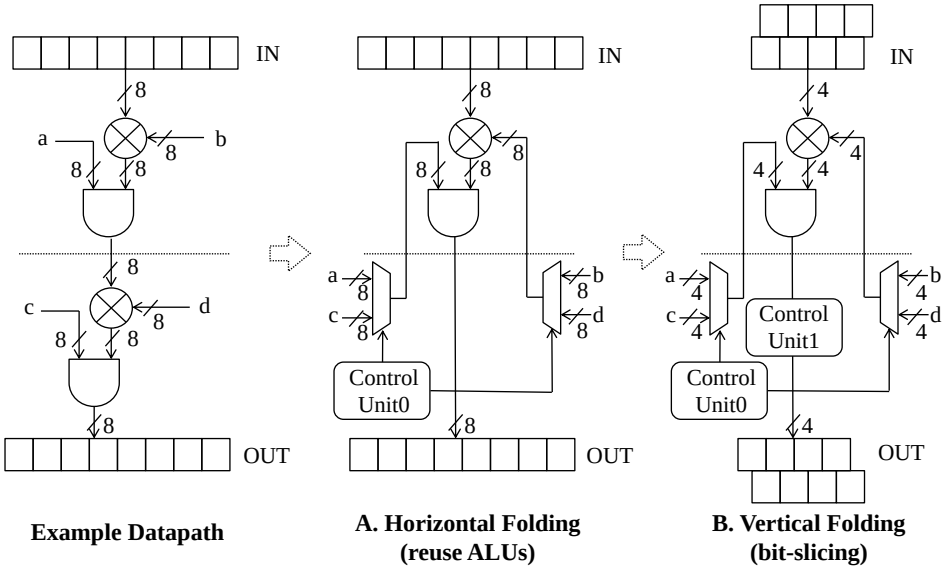


Figure 5.11: The two techniques of datapath folding.

will also cause control overhead.

There is no universal answer on how to select the horizontal or vertical or combined folding for the optimal architectural design. The choice has to be made by a tradeoff between ALU complexity and control unit cost. However, we recommend that lightweight hash in general may benefit more from vertical folding (or bit-slicing). One of the key reasons is that the bit-slicing will also change the memory structure as shown in Fig. 5.11. This may greatly affect the optimization at the lower silicon implementation level, as will be shown through the Cubehash design further in this work.

Silicon Implementation Optimization: Memory Structures

The importance of memory design was recently identified in the new lightweight block cipher designs [32]. The authors found that most of existing lightweight block cipher implementations not only differ in the basic gate technology, but also in the memory efficiency, namely the number of GEs required for storing a bit. Most of the lightweight block ciphers have more than 50% of the total area dedicated for memory elements, mostly flip-flops (FFs), and a single flip-flop in UMC 130nm standard-cell library ($fsc0l\_d\_sc\_tc$) may cost 5 to 12

Table 5.5: Compare the characteristics of different memory structures in typical ASIC standard-cell technologies [137]. (*: the size parameters are for Artisan standard library 130*nm* SRAM and Single-Port Register File generation.)

	Flip-Flops	SRAM	Register File
Density	Low	High	Medium High
Power	High	Medium	Low
Speed	Low	High	Medium
Latency	None	1 clock cycle	1 clock cycle
*Size Range	No limit	512-bit (256 words×2-bit) to 512 kbits	64-bit (8 words×8-bit) to 32 kbits
Application	Very small size memory	Large size memory	Small-to-Medium size memory

GEs depending on the different types of FFs in use.

Compared with lightweight block ciphers, the memory requirement for a hash is much more demanding as hash functions usually have a much larger state size (e.g. typically 512-bit or 1024-bit). However, in most published lightweight hash implementations we can hardly find discussions on optimization of memory structures. In contrast, we claim that lightweight hash implementations can benefit even more from careful memory design.

There are generally three types of read/write memory in ASIC standard-cell technologies shown in Table 5.5. Flip-flops, SRAMs, and register files each represent a different configuration of memory cells. They each have a different density (GEs/bit) and power dissipation. In addition, SRAMs and register files typically require a minimum size, defined by the technology library used. Considering our earlier discussion on architecture folding, register files and SRAMs obviously will be very useful to support vertical folding.

To summarize the selection of memory structures, for lightweight hash designs Register File (RF) is the ideal memory structure if the required memory size fits in the applicable range and configurations. Fig. 5.12 represents our key observations so far.

If an algorithm has limited state size, and a register file cannot be used, then lightweight hardware implementation is mainly about the tradeoff between datapath folding and control

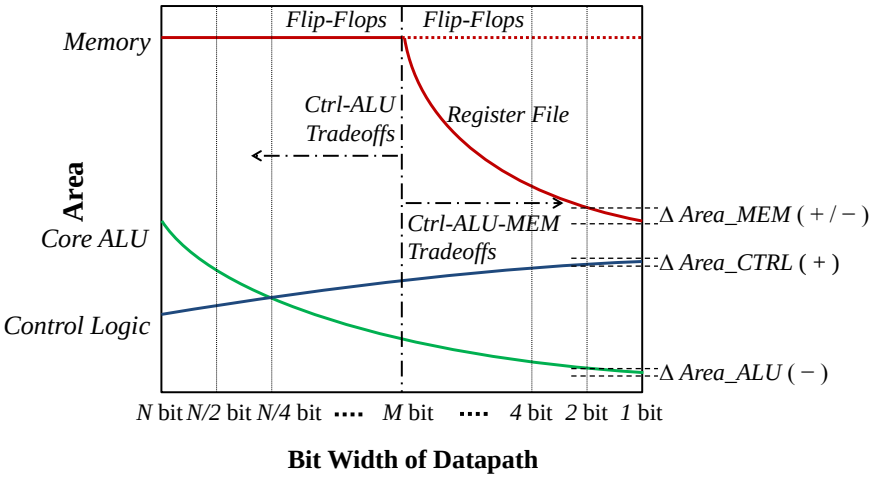


Figure 5.12: The cost model for lightweight hash designs.

overhead. However, as shown in Fig. 5.12, if an implementation can use a register file, then the cost of storage should be included in the area tradeoffs. Also, we will show that the impact of low-level memory implementation can be substantial, and that it can affect design decisions.

From the hardware engineers' perspective, following the curves in Fig. 5.12, one can make optimization decisions according to different hash specifications at different levels. If low cost is the primary target, one may also follow this graph to check whether your design is at the optimal point. By fully analyzing the interactions between bit-slicing and register file configurations under a target ASIC technology, one may gain a better understanding of when bit-slicing will become meaningful.

5.2.5 ASIC Library Dependent Cost Analysis of Quark

In this section, we investigate the impact of technology library selection on the overall GE count of a design. We do this through the example of the Quark lightweight hash function.

Table 5.6: Summary of technology nodes and standard-cell libraries used in technology dependent cost analysis.

Technology	Vendor	Standard-Cell Library	Notes
90 *nm*	UMC	*fsd0a_a_generic_core_tc*	Standard Performance Low-K
130 *nm*	UMC	*fsc0g_d_sc_tc*	Standard Performance High Density
180 *nm*	UMC	*fsa0a_c_sc_tc*	High Performance
130 *nm*	UMC	*fsc0g_d_sc_tc*	Standard Performance High Density
130 *nm*	IBM	*scx3_cmos8rf_rvt_tt_1p2v_25c*	Standard Performance
130 *nm*	IBM	*scx3_cmos8rf_lpvt_tt_1p2v_25c*	Low Power

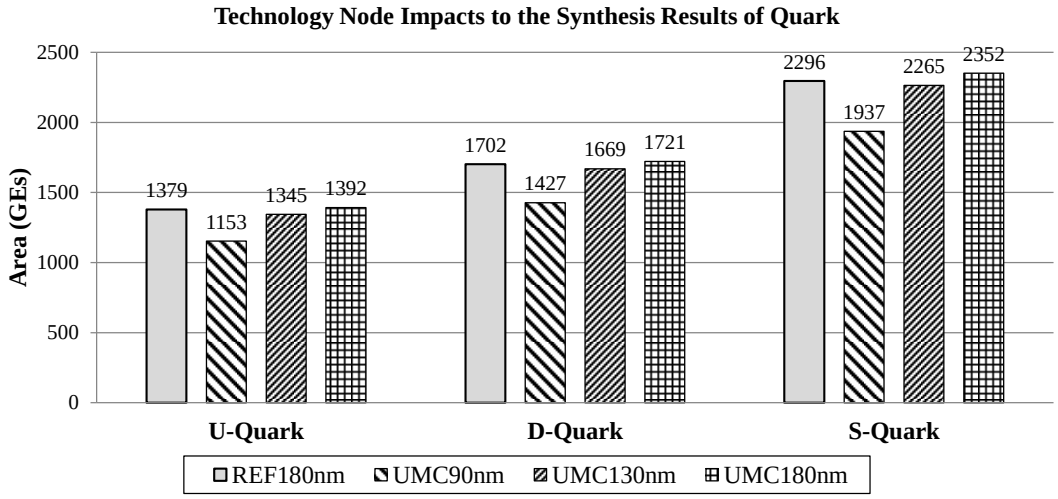


Figure 5.13: Cross-comparison of Quark post-synthesis area at different technology nodes. (Note: numbers of REF180*nm* refer to the hardware cost presented in [8])

Overview of Quark

The Quark hash family by Aumasson *et al.* was presented at CHES2010 [8], using sponge functions as domain extension algorithm, and an internal permutation inspired from the stream-cipher GRAIN and the block-cipher KATAN. There are three instances of Quark: U-Quark, D-Quark and S-Quark, providing at least 64-, 80-, 112-bit security against all attacks (collisions, second preimages, length extension, multi-collisions, etc.) [9]. The authors reported the hardware cost after layout of each variant as 1379, 1702 and 2296 GEs at 180*nm*.

The open-source RTL codes found at the Quark website [9] help us evaluate the standard-cell library impact on the hardware cost by reusing the same source codes. The evalua-

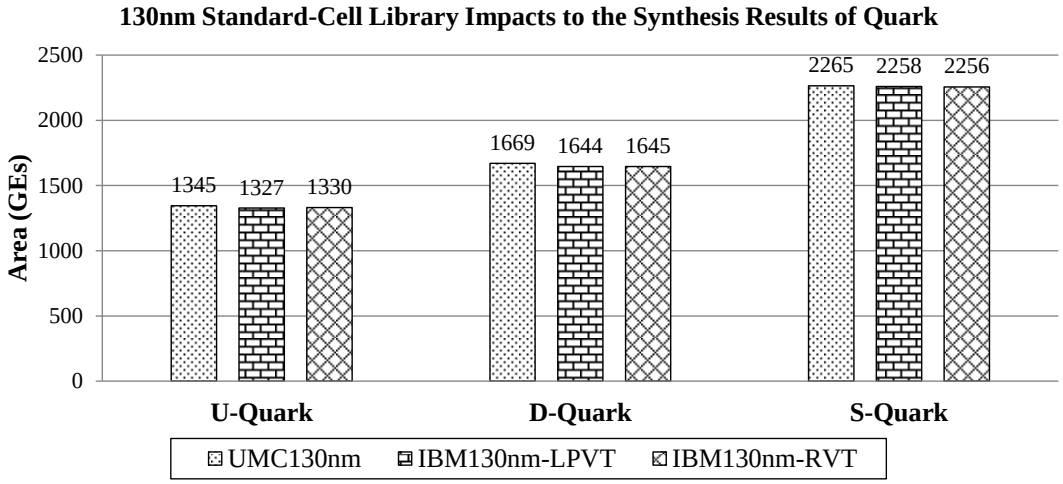


Figure 5.14: Cross-comparison of Quark post-synthesis area with different 130nm CMOS standard-cell libraries.

tion is performed at two steps: first, we look at cost variations at three technology nodes (180nm/130nm/90nm); second, at the same 130nm technology node we have tried: a) two standard-cell libraries from different vendors; and b) two different standard-cell libraries from the same vendor. A list of all the technology nodes and libraries can be found at Table 5.6.

The Impact of Technology Nodes and Standard-Cell Libraries

To evaluate the standard-cell library impacts we used the Synopsys Design Compiler (C-2009.06-SP3) to map the Quark VHDL source codes [9] to all the different technologies and libraries listed in Table 5.6. The synthesis constraints are set to optimize for the minimum area.

Take the smallest U-Quark as an example. As shown in Fig. 5.13, the synthesized circuit area varies from 1153 GEs to 1392 GEs in 90nm and 180nm, respectively. The differential area, 239 GEs, implies a variation range from -17.2% to 20.7%. The similar trend can also be found for D-Quark and S-Quark. In average, the area variation is from -17.3% to 20.9% for all the three Quark variants.

Through the above exploration we have demonstrated that a direct comparison of circuit

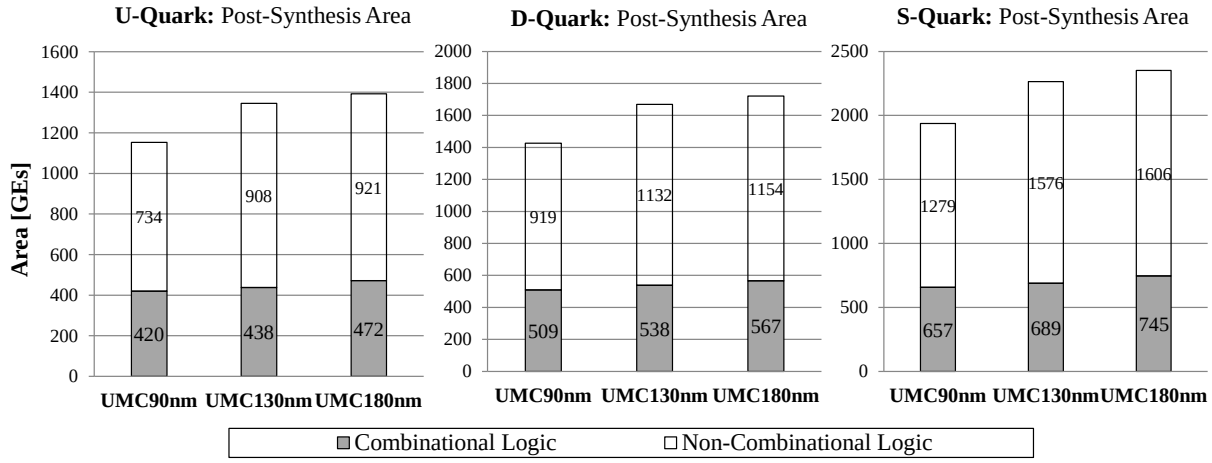


Figure 5.15: Technology impact on the combinational logic *vs.* non-combinational logic after synthesis.

area obtained at different technology nodes may cause significant inaccuracy. However, as shown in Fig. 5.14 comparing synthesis results using different libraries (UMC *vs.* IBM and Low Power Vt Library *vs.* Regular Vt Library) at the same $130nm$ technology node shows very small variations with an acceptable variation range from -1.1% to 1.1%. This means more accurate comparison in hardware cost could be achieved if the same standard-cell libraries are used or at least different standard-cell libraries are at the same technology node.

5.2.6 Causes of Area Variations

From the above exploration of the Quark synthesized results, we can observe that the technology selection has a much larger impact than the standard-cell library selection at the same technology node (up to 20.9% *vs.* 1.1%). To more precisely interpret the results and understand the differences, we look further into all the standard-cell libraries listed in Table 5.6 and the corresponding post-synthesis netlists.

The Quark RTL designs have very simple logic operations in their compression function and most of the circuit area is used for storing the state bits. For a further analysis of

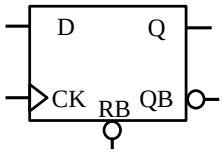
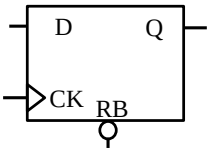
	D Flip-Flop w/ Clear	D Flip-Flop w/ Clear & Single-Output
		
UMC90nm	19*DFFRBX1(4.75GEs)	143*QDFFRBX1(4.50GEs)
UMC130nm	33*DFFRBEGD(6.00GEs)	129*QDFFRBCGD(5.50GEs)
UMC180nm	4*DFFRBN(6.33GEs)	158*QDFFRBN(5.67GEs)
IBM130nm-RVT	162*DFFRX2TF(5.75GEs)	N/A
IBM130nm-LPVT	162*DFFRX2TS(5.75GEs)	N/A

Figure 5.16: The distribution of different register types in the U-Quark netlists using different standard-cell libraries.

which part of the design contributes more to the variations across different technologies, we show the portion of combinational and non-combinational logic of each synthesized design in Fig. 5.15. From this graph, we can clearly see that the combinational part (grey portion), largely determined by the logic complexity of the compression functions, has much less variations under different technology nodes and contributes an average of 20.9% to the total area variation for all cases. The non-combinational part (white portion), mainly occupied by the storage of state bits, is more sensitive to different technology nodes.

To further analyze the non-combinational part of U-Quark, we compare the RTL design source codes with the post-synthesis netlist. Out of the total 162-bit storage elements, 136-bit are used for storing the hash state and the rest 24-bit are for the control path, including counter registers and state control bits. When these storage elements are synthesized and mapped into the netlist, two types of register are automatically instantiated, D Flip-Flop (DFF) with Clear and DFF with Clear & Single-Output. The only difference between these two types of DFF is the extra inverter to generate QB. As shown in Fig. 5.16, this difference brings different cost overhead in different technologies, and the synthesis tool will select different DFFs types to achieve minimum area according to the area efficiency of different standard cells. And that is why we can see different distributions of the register type selection.

For the relatively large variation (up to 20.9%) between different technology nodes, we can clearly see in the Fig. 5.16 that the area efficiency (in terms of GEs/DFF) of DFF is the determining factor of the 78.2% variation contributed by the non-combinational part of the U-Quark design. For the 162-bit storage in U-Quark, the area efficiency of DFF has a variation up to 25.5% across UMC $90nm$, $130nm$ and $180nm$, respectively.

For the small variation (up to 1.1%) between different standard-cell libraries at the same $130nm$ technology node, this can be explained by the close area efficiency of DFF with UMC$130nm$ and IBM$130nm$-RVT/LPVT libraries, which are 5.60 GEs/DFF (in average) and 5.75 GEs/DFF with less than 2.7% variations.

5.2.7 Storage Structure Dependent Cost Analysis of CubeHash

In this Section we discuss the impact of storage architecture on cost, through the example of a bitsliced CubeHash design proposed by Bernstein [19–21].

Overview of CubeHash

CubeHash is among the 14 NIST SHA-3 Second Round candidates. Although it was not selected to be considered as one of the SHA-3 finalists, CubeHash attracted extensive third-party analysis and no practical attacks were found. CubeHash is also an exception among Second Round SHA-3 candidates in terms of hardware footprint. The smallest implementation of CubeHash was reported with 7630 GEs in 130 nm technology [17]. Because CubeHash offers a high security level with a very simple structure, it is worthwhile to look further into the lightweight implementation of this design.

CubeHash is a sponge-like hash algorithm that is based on the iteration of a fixed permutation, T, on a state of 128 bytes. A simple padding rule is applied to the input message. The algorithm consists of three stages: initialization, message injection, and finalization. In addition to the parameter, h, for the bit length of the message digest size, there are four

tunable parameters: 1) the number of rounds of message injection, r; 2) the message word size, b, in bytes; 3) the initialization rounds, i; and 4) the finalization rounds, f. Thus, for the message injection stage of CubeHash$i + r/b + f - h$, b bytes of the message are XORed into the state before r iterations of T. The initialization and finalization stages each consist of i and f rounds of T on the state, respectively. Initialization may either be pre-computed to save time, or computed on-the-fly to save memory. Finalization is preceded by the flipping of a single bit of the state. No round constants or length indicators are used, other than the encodings of the parameters in the three initial-state words.

In this work we evaluated the CubeHash512 (Version November 2010), defined as CubeHash 16+16/32+32−512. The security bits for preimages attacks is 384 and 256 for collision attacks. The operations in the round permutation, T, are modular additions, XORs, and rotations, which are very suitable for bit-sliced implementations.

VLSI Implementations

In Section 5.2.4: Lightweight Hash Implementation Methodology, we describe two general techniques, bit-slicing and memory structures, for lightweight hash designs. In this section, we will show how to use them in optimizing a new hash proposal, CubeHash [22], for lightweight implementations. By using CubeHash as a case study, we also show that with the combination of bit-slicing and register file as the main storage, CubeHash can also be a ideal lightweight hash candidate (less than 6,000 GEs with 130nm technology) but with much higher security levels. This makes Cubehash complementary to other new lightweight hash proposals with lower security margins. For practical aspects, we will also show the interaction between bit-slicing and register file configurations, and how this interaction will affect the hardware cost.

The architecture of the original CubeHash with 32-bit datapath is shown in Fig. 5.17. The 32-bit datapath, as shown, can finish one round of CubeHash core function in one clock cycle. The left and right part of the datapath are almost identical except for the different

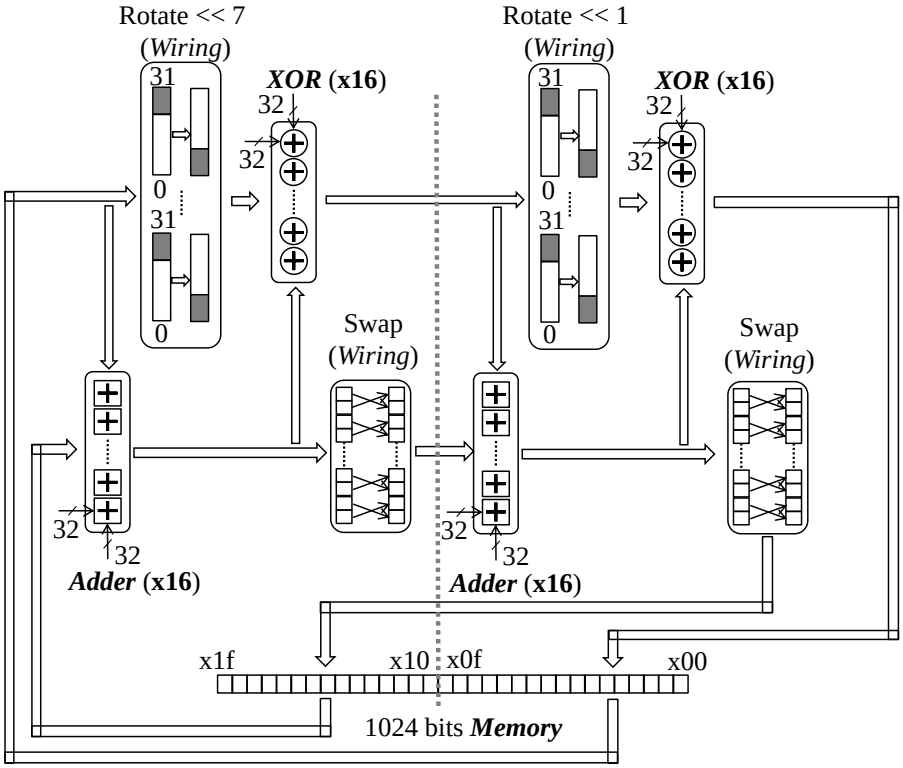


Figure 5.17: The hardware architecture of CubeHash with 32-bit datapath.

rotations. Since rotation and swap operations can be realized through simple wiring in hardware without logics, the ALUs of datapath mainly consists of 32-bit adders and 32-bit XORs. The CubeHash state with the configuration of 1024-bit $\times$ 1 word (1024b1w) can only be implemented using flip-flops.

For the one-cycle-per-round CubeHash architecture, denoted as Cube1, the 32-bit datapath and 1,024-bit flip-flops will consume most of the circuit area. So, we can apply the bit-slicing technique to reduce the datapath bit width. For bit-sliced ALUs of CubeHash datapath, we only need to consider the 32-bit adders. Here the ripple carry adder with full adders can be used. Although rotation and swap operations are free in hardware, for different degrees of bit-slicing preparing the corresponding data format and vector ordering needs to be carefully addressed. More details on how to map these operations in 2, 4, 8 bit-sliced versions of CubeHash can be found at [19–21].

We have implemented two bit-sliced versions of CubeHash with a 2-bit and a 4-bit datapath, respectively. We denote these two versions as Cube32 and Cube16. As a result of the bit-slicing, the CubeHash state memory configuration changes from the default '1024b1w' to '64b16w' and '128b8w', respectively.

We chose to define our design space only with these two bit-sliced versions for two reasons. First, the register file needs to have at least 8 fields, so that bitslicing above 4-bit would imply storage overhead. Second, after analyzing the results of Cube32 and Cube16, we found the throughput of the bit-serial implementation of CubeHash to be too slow to be competitive. This will be illustrated further.

Cost Analysis of Bit-Sliced CubeHash Designs

The overall cost estimation process can be split into basic three categories:

- Datapath logic
- Control logic

Table 5.7: The comparison of the bit-sliced datapath logic composition and estimated area reduction. (Note: the 'ADDER' type is carry ripple adder.)

Datapath Logic	Cube16	Cube32
# of 4-bit XOR [8.00GEs]	24	0
# of 2-bit XOR [4.00GEs]	0	24
# of 4-bit ADDER [22.00GEs]	16	0
# of 2-bit ADDER [16.25GEs]	0	16
# of 4-bit MUX [7.75GEs]	64	0
# of 2-bit MUX [4.25GEs]	48	64
# of 1-bit MUX [2.50GEs]	0	48
Total [GEs]:	1244	748

- Memory

Take the CubeHash design as an example, identifying datapath logic is relatively easier from architecture block diagrams shown in Fig. 5.17 or C reference models [19–21]. Moving from Cube16 (4-bit datapath) to Cube32 (2-bit datapath), the estimated cost reduction in the datapath logic is 469 GEs as illustrated in Table 5.7.

However, if we look at the total cost reduction in the datapath and control logic (excluding the 1024-bit memory), the combinational part reduction is around 457 GEs and non-combinational part increase is around 188 GEs with a total cost reduction of 268 GEs. Since the estimated cost reduction of 469 GEs in datapath logic is for combinational part, the result of the overall cost reduction means around 200 GEs control logic overhead is observed when future reducing the datapath from 4-bit to 2-bit. And the overhead is mainly composed of non-combinational logic, which is largely caused by increased number of temp registers for storing the intermediate results and increased round counter size.

When optimizing the bit-sliced implementations of Cube32 and Cube16, we also have the option to implement memories with single-port (SP) or dual-port (DP). For single-port memories, since they have single set of address and controls with single access (read/write)

Table 5.8: The comparison of different configurations of flip-flops based memory (register arrays).

Configuration	Area [GEs]	Efficiency [GEs/bit]	Ratio
64b16w-SP	7334	7.2	70%
64b16w-DP	7700	7.5	72%
128b8w-SP	7192	7.0	68%
128b8w-DP	7953	7.8	70%

to the memory, it will save some area due to the simpler peripheral circuits compared with dual-port memories with the same size. However, the CubeHash throughput will cut into half with SP-memories because in this case read and write operations need to be in separate clock cycles.

A common way used in most existing hash designs to implement the memory module (hash state storage) is to let the synthesis tools directly compile it into flip-flops based register arrays. As shown in Table 5.8, the average memory efficiency of different configurations of the register array is 7.4 GEs/bit and ratio of memory in the overall design cost is above 70%. This means previous optimizations on minimizing the datapath logic and control logic may only affect a small portion of the overall circuit area, which is less than 30%, and lightweight hash implementation may benefit more from optimizing the storage structure.

Quantify the Impact of Storage Structure

To further optimize the storage structure of the bit-sliced implementations of Cube32 and Cube16, we also have the option to implement flip-flops or register file based memories. A summary of different characteristics of flip-flops, register file and SRAM based memory options can be found in Table 5.5 of Section 5.2.4.

We emphasize that after choosing single- or dual-port memory types in hardware architecture design, the RTL code for a flip-flop based version and for a register-file based version

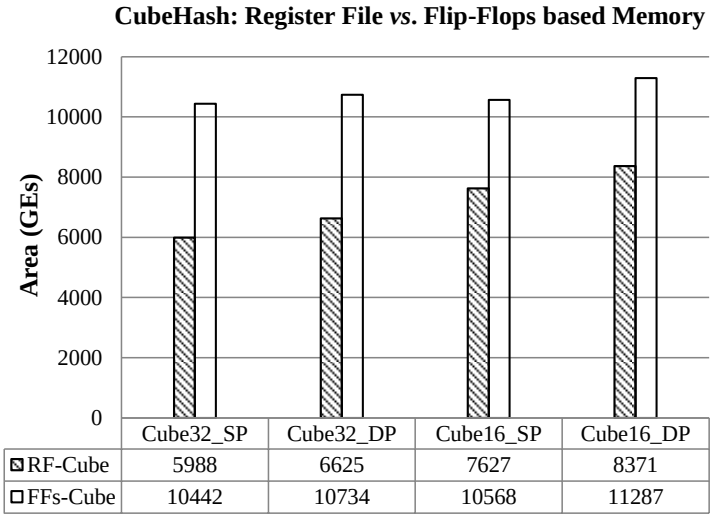


Figure 5.18: Comparison of the impact of different memory types to the CubeHash area.(Note: the latency of Cube32_SP, Cube32_DP, Cube16_SP and Cube16_DP are 1024, 512, 512 and 256 cycles, respectively)

is identical. Both of them share the same standard memory interface. However, after using the same synthesis constraints targeting smallest area to synthesize both of the designs with IBM MOSIS 130nm low power standard-cell library and register file modules from Artisan Standard Library 130nm Register File Generator, the advantages of using register file becomes obvious, as shown in Fig. 5.18.

The RF-based CubeHash designs can save between 26% and 43% of the area over flip-flop based ones. As we look further into the memory efficiency metric in terms of GEs/bit, for the size of 1,024 bits the flip-flops based memories is almost constant with small variations between 7.0 GEs/bit and 7.8 GEs/bit; however, the register file based ones have the densest configuration of 32b32w_SP with highest efficiency at 2.3 GEs/bit and 4.9 GEs/bit for the lowest efficiency with 128b8w_DP configuration as shown in Fig. 5.19.

After analyzing the area composition of RF-based CubeHash designs shown in Fig. 5.20, by reducing the datapath from 4-bit to 2-bit, we only save less than 300 GEs in average, which is less than 5% of the total CubeHash area; however, moving from Cube16_DP to Cube32_SP we can save 2,156 GEs, which is 26% area reduction of the overall Cube16_DP

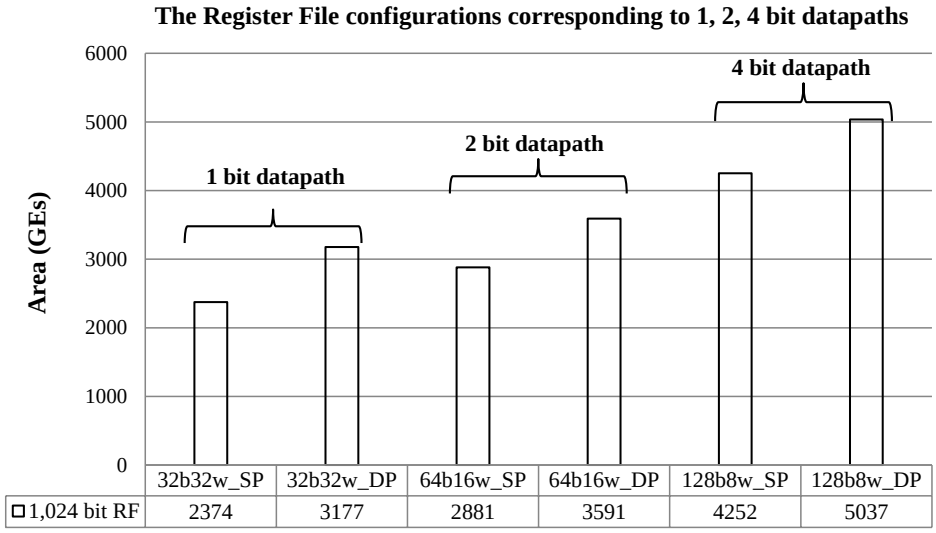


Figure 5.19: Comparison of the cost of different configurations of the register file. (Note: for all single-port memories the read and write operations need to be in separate clock cycles)

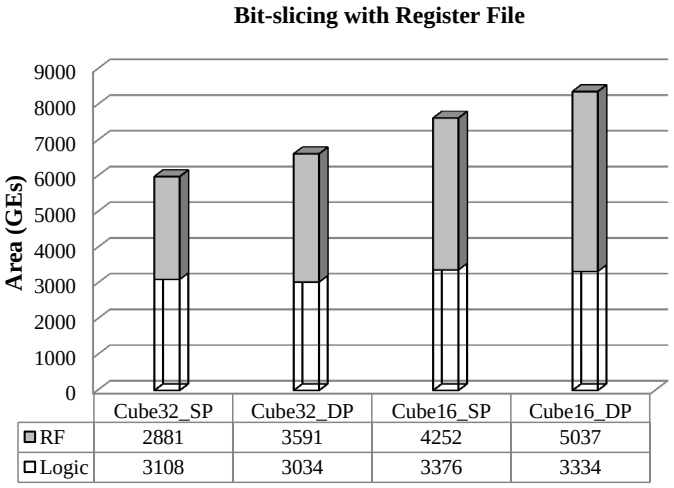


Figure 5.20: Comparison of the storage ratio in different memory configurations.

area.

This interesting comparison delivers an important message: *bit-slicing may not provide a reduction in datapath area.* In fact, in an extreme case (e.g. bit-serial implementation), bit-slicing may even increase the logic area with more incurred control overhead than the reduced area in ALUs; however, the associated changes in memory configurations may have greater impact on the overall area reduction.

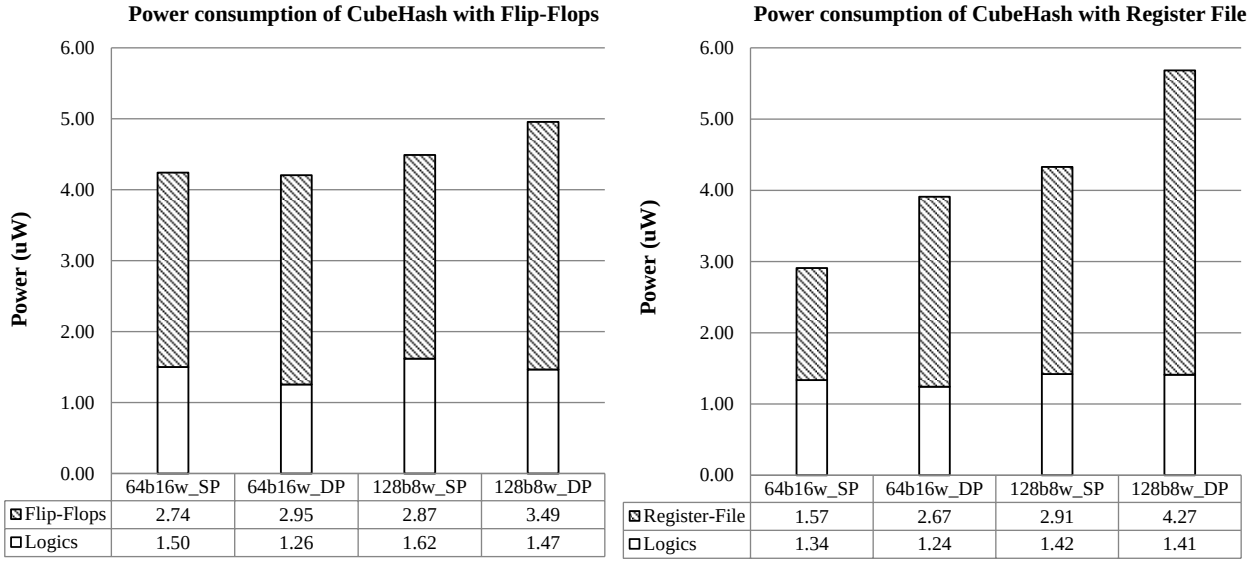
For the bit-serial CubeHash design, according to the above analysis the datapath logic area reduction will become negligible. Using register file with 32b32w_SP configuration can save 500 GEs, but this will make the latency as bad as 2048 cycles, which even cannot meet the RFID tag's requirement of 1800 cycles [115]; while for the other configuration with 32b32w_DP, with the same latency of 1024 cycles as Cube32-64b16w_SP the area of the register file will increase (possibly due to the memory shape). Therefore, bit-serial version of CubeHash is excluded in our design space.

We have also performed post-synthesis simulation and compared the power efficiency of flip-flops and register file based CubeHash designs. As we can see from Fig. 5.21, register file based designs are in general more power efficient than the flip-flop based ones. For all the cases, the power consumption at 100 KHz are way below the RFID's required power budget of 27 μW [115].

To compare lightweight CubeHash implementations with other published work, CubeHash can offer much higher security levels with satisfying throughput at an acceptable cost. Based on the lightweight hash design methodology proposed in this work, we have shown how to optimize a general purpose hash algorithm for lightweight applications. With the given security and cost, we think CubeHash is an ideal complement to other existing lightweight proposals. As also shown in Table 5.9, all these lightweight hash designs take advantages of bit-slicing but only our design makes a better usage of the memory structure.

Table 5.9: Comparison of the characteristics of different lightweight hash designs. (Note: 1) 'Dig.', 'Dp', 'Proc.' and 'Thr.' refer to digest size, datapath width, process and throughput, respectively; 2) throughput and power numbers are based on designs running at 100KHz; 3) a graphical presentation of selected metrics can be found in the Section 5.2.2 for easier comparison)

Hash Function	Pre. [bit]	Coll. [bit]	Dig. [bit]	Latency [cycles]	Dp [bit]	Proc. [nm]	Area [GEs]	Thr. [$kbps$]	Power [μW]
DM-PRESENT-80 [29]	64	32	64	547	4	180	1600	14.63	1.83
DM-PRESENT-128 [29]	64	32	64	559	4	180	1886	22.90	2.94
PHOTON-80/20/16 [58]	64	40	80	708	4	180	865	2.82	1.59
PHOTON-128/16/16 [58]	112	64	128	996	4	180	1122	1.61	2.29
H-PRESENT-128 [29]	128	64	128	559	8	180	2330	11.45	6.44
U-Quark [8]	128	64	128	544	1	180	1379	1.47	2.44
ARMADILLO2-B [12]	128	64	128	256	1	180	4353	25.00	–
PHOTON-160/36/36 [58]	128	80	160	1332	4	180	1396	2.70	2.74
D-Quark [8]	160	80	160	704	1	180	1702	2.27	3.10
ARMADILLO2-C [12]	160	80	160	320	1	180	5406	25.00	–
C-PRESENT-192 [29]	192	96	192	3338	12	180	4600	1.90	–
PHOTON-224/32/32 [58]	192	112	224	1716	4	180	1736	1.86	4.01
S-Quark [8]	224	112	224	1024	1	180	2296	3.13	4.35
PHOTON-256/32/32 [58]	224	128	256	996	8	180	2177	3.21	4.55
ARMADILLO2-E [12]	256	128	256	512	1	180	8653	25.00	–
Keccak-f[200] [86]	64	32	64	900	8	130	2520	8.00	5.60
SPONGENT-88 [28]	80	40	88	990	4	130	738	0.81	1.57
SPONGENT-128 [28]	120	64	128	2380	4	130	1060	0.34	2.20
Keccak-f[400] [86]	128	64	128	1000	16	130	5090	14.40	11.50
SPONGENT-160 [28]	144	80	160	3960	4	130	1329	0.40	2.85
SHA-1 [115]	160	80	160	344	8	130	5527	148.88	2.32
SPONGENT-224 [28]	208	112	224	7200	4	130	1728	0.22	3.73
SPONGENT-256 [28]	240	128	256	9520	4	130	1950	0.17	4.21
Cube8/1-512 [17]	384	256	512	512	32	130	7630	2.00	–
Cube32_SP-RF	**384**	**256**	**512**	**1024**	**2**	**130**	**5988**	**25.00**	**2.91**
Cube32_DP-RF	**384**	**256**	**512**	**512**	**2**	**130**	**6625**	**50.00**	**3.91**
Cube16_SP-RF	**384**	**256**	**512**	**512**	**4**	**130**	**7627**	**50.00**	**4.33**
Cube16_DP-RF	**384**	**256**	**512**	**256**	**4**	**130**	**8371**	**100.00**	**5.68**



	64b16w_SP	64b16w_DP	128b8w_SP	128b8w_DP
Flip-Flops	2.74	2.95	2.87	3.49
Logics	1.50	1.26	1.62	1.47

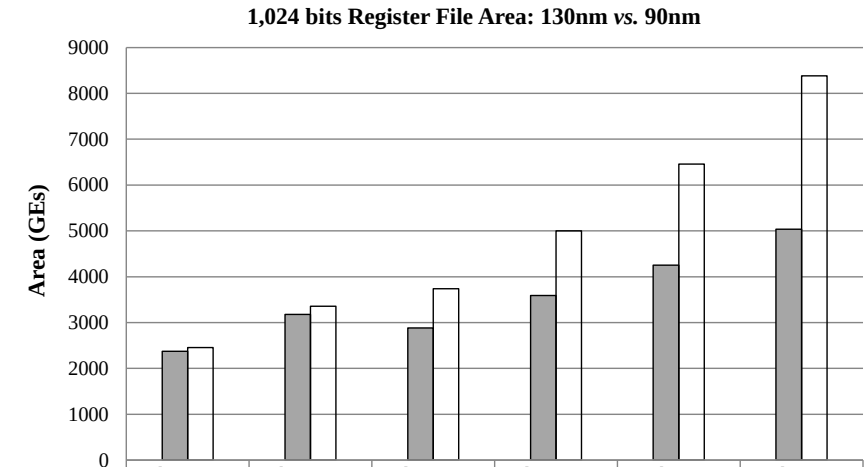
	64b16w_SP	64b16w_DP	128b8w_SP	128b8w_DP
Register-File	1.57	2.67	2.91	4.27
Logics	1.34	1.24	1.42	1.41

Figure 5.21: Comparison of the power consumption with flip-flops and register file based CubeHash implementations.

5.2.8 Considerations with Using Register File

The above two case studies show why the technology can affect the lightweight hash implementation results through quantitative cost analysis, and demonstrate how to make use of the different technology options (e.g. different types of memory) to further reduce the storage portion of the circuit area. In this Section, we discuss several issues with adopting different types of memory in the backend design and optimization of other lightweight cryptographic algorithms.

Similar as the Quark case study, the impact of technology nodes on the area of register file can also be significant. Here we use the register files generated from UMC 90nm (fsd0a_a) as a comparison with those from IBM 130nm (scx3_cmos8rf_lpvt). As illustrated in Fig. 5.22, for some configurations (e.g. 32b32w_SP/DP) the memory efficiency is very close; however, for the configuration of 128b8w, the variation of memory efficiency can be up to 66.41%. This means the effectiveness of using register file to replace the flip-flops based register array for area reduction may differ a lot at different technology nodes with different memory compilers



	32b32w_SP	32b32w_DP	64b16w_SP	64b16w_DP	128b8w_SP	128b8w_DP
IBM130nm	2374	3177	2881	3591	4252	5037
UMC90nm	2455	3358	3740	5002	6458	8382

Figure 5.22: Comparison of the impact of different technology nodes to the register file area.

from vendors. Therefore, when reporting the circuit area as a result of using register file as the storage option, it is necessary to give more details of the memory IP information and the memory efficiency metric as a reference.

5.2.9 Summary

Recent lightweight hash proposals have presented a tradeoff between security and cost; cryptographic engineers, on the other hand, have proposed more fine-grained optimizations to achieve the most efficient implementations. By quantifying technology impacts to the cost analysis of different lightweight hash implementations, this work shows the benefits of making these two groups people working in an interactive design process and at different abstraction levels. Our technology dependent cost analysis may help cryptographic engineers have better presentation of the metrics and avoid some common pitfalls. The proposed lightweight hash design methodology establish the link between algorithm design and silicon implementation with a strong emphasis on the interaction between hardware architecture and silicon implementation. The cost model of lightweight hash designs reveals the interaction between bit-slicing and memory structures may divide the design space for lightweight implementa-

tion into two regions: one is mainly about the tradeoff between datapath folding and control overhead, and the other one needs to add the low-level memory structure as an additional tradeoff point.

5.3 Conclusion

In this chapter we discuss two technology dependent issues in the evaluation of cryptographic hardware.

The first issue is how to perform cross-platform comparison between the FPGA and ASIC results. Our research shows that ASIC and FPGA designers may come to different conclusions when it comes to making a statement on the most efficient SHA-3 candidate in hardware. However, each of ASIC and FPGA SHA-3 designs offer a similar design space (tradeoffs of around 7 times between most and least efficient ones in both area and power metrics as shown in Figure 5.7). We also provide some insights on how to look at SHA-3 hardware benchmarking results in different platforms. In cases where the platform is already fixed (ASICs or FPGAs), one should exclusively rely on FPGA-specific or ASIC-specific benchmarks, depending on the chosen platform. Conclusions on ASIC implementations based on FPGA results, or vice versa, will almost certainly be inaccurate. In some other cases, where you are looking to understand the SHA-3 candidates and where you do not yet have chosen a platform, it will be equally interesting to compare both the ASIC and FPGA SHA-3 results, because they point out different aspects of SHA-3 hardware implementations.

The second issue is how to optimize hash functions for lightweight implementations. Our technology dependent cost analysis may help cryptographic engineers have better presentation of the metrics and avoid some common pitfalls. The proposed lightweight hash design methodology establish the link between algorithm design and silicon implementation with a strong emphasis on the interaction between hardware architecture and silicon implementation. The cost model of lightweight hash designs reveals the interaction between bit-slicing

and memory structures may divide the design space for lightweight implementation into two regions: one is mainly about the tradeoff between datapath folding and control overhead, and the other one needs to add the low-level memory structure as an additional tradeoff point.

Chapter 6

Secure Implementations of ECC Cryptosystems

Several kinds of implementation attacks on cryptographic devices have been published. They can be categorized into two types: passive attacks and active attacks [7]. Passive attacks are based on the observation of side-channel information such as the power consumption of the chip or its electromagnetic emanations. Examples of passive attacks include SPA, DPA, SEMA and DEMA attacks. On the other hand, active attacks, including fault injection attacks, deliberately introduce abnormal behavior in the chip in order to recover internal secret data. As mentioned earlier, a cryptosystem will fail at its weakest link and one countermeasure against one SCA attack may benefit another attack. Therefore, when integrating countermeasures into cryptosystems, it is crucial to consider active as well as passive attacks, and select a set of countermeasures that resists a collection of passive and active attacks.

This chapter presents a comprehensive analysis of the existing SCA attacks and fault attacks on ECC cryptosystem implementations and demonstrate how to select and integrate a set of countermeasures to resist a collection of attacks.

6.1 Introduction

Traditional cryptanalysis assumes that an adversary only has access to input and output pairs, but has no knowledge about internal states of the device. However, the advent of SCA showed that a cryptographic device can leak critical information. By monitoring the timing, power consumption, EM emission of the device or by inserting faults, adversaries can gain information about internal data or operations and extract the key out of the cryptographic device without mathematically breaking the primitives.

Performance, security and cost are the three important dimensions of ECC implementations. ECC accelerators should support multiple security and performance levels, allowing the system to adjust its security-performance to application-specific needs [6]. To achieve these goals, much research has been conducted targeting different aspects, and most research topics fall into two categories: efficient implementations and security analysis.

For security analysis, ECC implementations are known to be vulnerable to various SCA attacks, including PA, EMA and FA attacks. Since Kocher *et al.* [94] showed the first successful PA attacks, there have been dozens of proposals for new SCA attacks and countermeasures. These attacks and countermeasures all tend to concentrate on a single abstraction level at a time [123]. For example, the Smart Card software is developed on fixed hardware platforms, so the results in that area are software-based solutions. At the same time, many special circuit styles [79] have been developed to address PA at the hardware level. Such circuit level solutions are treated independently from the software-level solutions.

From the above descriptions, two gaps can be found in current ECC research. First, security has been generally treated as a separate dimension in designs and few researchers have proposed countermeasures targeting at system integration. For example, some of the FA attack countermeasures or fault detection methods are just like software patches applied to the original algorithms (e.g. perform Point Verification [25] or Coherence Check [41, 54] during computation). The fault model is hypothesized without considering how to introduce

faults in an actual platform. Further, the impact of circuit-level PA countermeasures on area, performance and power consumption in large designs remains unclear. Second, most published papers proposed their own attacks with corresponding countermeasures and very few researchers discussed countermeasures targeting multiple attacks. Since a cryptosystem will fail at its weakest link, it is not surprising to see how a given countermeasure can actually introduce a new weakness, and thus enable another attack [142]. Although there has been some effort to connect the PA and FA countermeasures [13], solutions at system integration level are unexplored.

Therefore the question now becomes how to fill both of the gaps in one system design. Specifically, we try to move these two research topics to the next step by building a flexible ECC coprocessor architecture with the ability to consider efficiency and security simultaneously and provide a unified countermeasure which can be easily integrated into system designs.

With new tampering methods and new attacks being continuously proposed and accumulated, designing a *secure* cryptosystem becomes increasingly difficult. While the adversary only needs to succeed in one out of many attack methods, the designers have to prevent all the applicable attacks simultaneously. Moreover, countermeasures of one attack may surprisingly benefit another attack. As a result, keeping abreast of the most recent developments in the field of implementation attacks and with the corresponding countermeasures is a never ending task.

In this research we provide a systematic overview of implementation attacks and countermeasures of ECC. However, this work has no intention to propose new attacks or new countermeasures. Instead, we describe several general principles for countermeasure selection. The survey in this dissertation can be used as a tool for selecting countermeasures in a first design iteration.

This survey has been influenced by Avanzi's report [10] and the books by Blake et al. [26] and Avanzi et al. [11]. All of them give an excellent overview of SCA attacks on ECC and

HECC up to their point of publication. This work, however, differs from previous work in three aspects. First, it includes recently reported attacks such as carry-based attack [49]. Second, we focus on the interaction of known attacks and countermeasures in a systematic way. Third, this survey proposes some guidelines for selecting countermeasures. Perfect countermeasures still do not exist.

6.2 Implementations and Physical attacks

Cryptographic transformations can be implemented in both software and hardware. While software implementations, running on general purpose microprocessors, are flexible and can be easily updated, hardware implementations, either on FPGAs or ASICs, can achieve higher performance.

Figure 6.1 shows the architecture of an ECC processor. Note that each component may refer to different types of realizations. For example, the ALU can be a standard ALU of a general purpose processor or a dedicated field multiplier. The temporary storage can be a RAM or a register file. A non-volatile memory, e.g. flash ROM, is normally used to store curve parameters.

An ECSM process starts with loading certain configurations (the definition of the curve, the underlying field, the coordinate system, the base point P) and the scalar k. While the base point P can be read either from the ROM or from outside, the scalar k is normally stored or generated inside the chip and should be protected. The output point, $Q = k \cdot P$, is not completely visible from outside. For example, El-Gamal decryption algorithm only returns the x-coordinate of Q.

In practice, execution of ECSM leaks information of k in many ways. Figure 6.1 also shows various SCA attacks on ECSM. We group the diversity of attacks into two main categories: passive and active attacks. Passive attacks do not require meddling with the device's input/outputs or working environment. Active attacks on the other hand try to

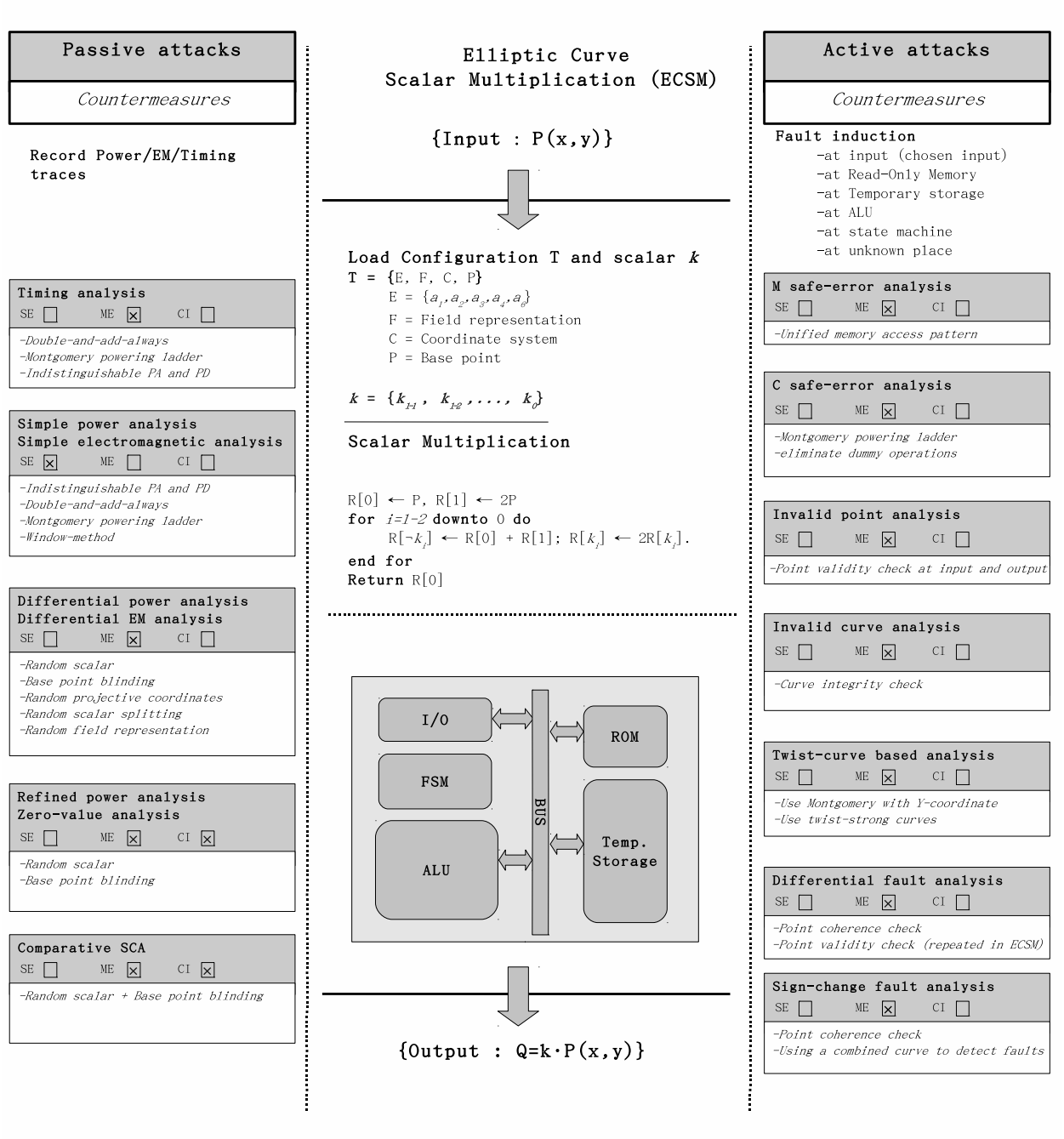


Figure 6.1: Elliptic curve processor architecture and related physical attacks. (SE = Single Execution, ME = Multiple Executions, CI = Chosen Input)

induce and exploit abnormal behavior in the device.

An important criterion to judge the cost of a specific SCA attack is how many executions are required to reveal the complete key stream. In Fig. 6.1, each attack is tagged with either SE (Single Execution) or ME (Multiple Executions). Another important criterion is that some attacks, such as doubling attack and refined power analysis, require the freedom of choosing the base point while some do not. The base point can be fixed or stored internally in some implementations, which makes attacks with this requirement significantly harder to mount.

In order to counteract passive SCA attacks at least one of the following links has to be removed:

1. The relation between the data or operations inside the device and the physical leakages (e.g. power traces, EM radiation traces, timing, etc.).

2. The relation between the hypothetical data and the actual data calculated in the device.

With respect to power analysis, there are two methods to achieve this: rendering the power consumption constant (e.g. using a special logic style [131]) or randomizing the intermediate data during the scalar multiplication.

In order to counteract fault attacks, two types of methods are used: error-detection and error-tolerance. The first method detects faults inserted in the elliptic curve parameters or the point multiplications. If faults are detected, the execution is aborted. The second method chooses an elliptic curve such that even if faults are inducted in the scalar multiplication, the adversary can not derive scalar from the faulty results. For example, twist-strong curves are error-tolerant under twist-curve attack.

6.3 Passive Attacks and Countermeasures

An adversary has a wide range of choice in attack strategies.

Timing Attacks and Simple SCA

Timing attacks exploit the timing variance with different inputs [93]. Careless implementations contain a vast number of sources of timing leakage. For example, timing variations can be caused by RAM cache or conditional branches. Although no papers have been published about a practical timing attack on ECC, many papers do mention the threat and provide the reader with suitable countermeasures.

Cryptographic implementations are vulnerable to simple power analysis attacks if the power traces show distinctive key-dependent patterns [94]. Alg. 1 shows the "double-and-add" algorithm for a point multiplication. The value of a key bit can be revealed if the adversary can tell the difference between point doubling and point addition from a power trace.

The *double-and-add-always* algorithm, introduced in [40], ensures that the sequence of operations to compute a scalar multiplication is independent of the value of the secret scalar through insertion of a dummy point additions. Another way to prevent simple SCA is making point addition and doubling indistinguishable. For example, dummy operations can be added at the field arithmetic level. This has the advantage of less overhead. On the other hand, the Hamming weight of the secret scalar might still leak.

Instead of making the group operations indistinguishable, one can rewrite them as sequences of side-channel atomic blocks that are indistinguishable for simple SCA [36]. Implementations based on the Montgomery ladder [85, 104, 109], shown as Alg. 3, are protected against timing attacks and simple SCA since the execution time of the scalar multiplication is inherently unrelated to the Hamming weight of the secret scalar.

Algorithm 2 Add-and-double-always point multiplication [40]

Input: $P \in E(\mathbb{F})$ and integer $k = \sum_{i=0}^{l-1} k_i 2^i$.
Output: kP.

1: $R[0] \leftarrow \mathcal{O}$.
2: **for** $i = l - 1$ downto 0 **do**
3: $\quad R[0] \leftarrow 2R[0]$, $R[1] \leftarrow R[0] + P$.
4: $\quad R[0] \leftarrow R[k_i]$.
5: **end for**

Return $R[0]$.

The last type of countermeasure is the usage of unified formulae for point doubling and addition [30]. Unified point addition formulae use a single formula to calculate both the doubling and the addition, resulting in an single sequence of operations for both.

Template Attacks

A template attack [33] requires access to a fully controllable device, and proceeds in two phases. In the first phase, the profiling phase, the attacker constructs a precise model of the wanted signal source, including a characterization of the noise. The second phase comprises the actual attack. Because of their reliance on data dependencies, template attacks exploit the so-called differential SCA type of leakage. The attack assumes that most of the side-channel information resides in the variance. So far not much research has been done on template attacks for public key algorithms. Medwed and Oswald [107] showed the feasibility of this type of attack on an implementation of the ECSDA algorithm. In [75] a template

Algorithm 3 Montgomery powering ladder [109]

Input: $P \in E(\mathbb{F})$ and integer $k = \sum_{i=0}^{l-1} k_i 2^i$.
Output: kP.

1: $R[0] \leftarrow P$, $R[1] \leftarrow 2P$.
2: **for** $i = l - 2$ downto 0 **do**
3: $\quad R[\neg k_i] \leftarrow R[0] + R[1]$, $R[k_i] \leftarrow 2R[k_i]$.
4: **end for**

Return $R[0]$.

attack on a masked Montgomery ladder implementation is presented. Template attacks, if feasible, are a major threat. Neither the double-and-add-always algorithm, nor blinding the scalar or base point resist template attacks. In fact, only randomizing the coordinates provides protection.

Differential SCA

Differential SCA attacks (DPA and DEMA) use statistical techniques to pry the secret information out of the measurements [94]. A differential attack adheres to a fixed working principle: a cryptographic device, supplied with the fixed secret key k, is sequentially fed with N input points P_i, $i \in \{1, 2, .., N\}$. During the encryption of input P_i under key k, a measurement over time of the side-channel $m_i(t)$ is recorded and stored. The attacker then chooses an intermediate value of the algorithm which depends both on the input point P_i and a small part of the secret key k. For each key candidate k' for the partial key and for each input point P_i, the attacker calculates the intermediate value and transforms it to a hypothetical leakage value $L_{k',i}$ with the aid of a hypothetical leakage model. For the correct key guess $k' = k$ there will be a correlation between the measurements $m_i(t)$ and the hypothetical leakages $L_{k,i}$ at some time instance t. This relation is uncovered by using statistical distinguishers such as a difference of means test, Pearson correlation or Spearman's rank correlation.

A straightforward countermeasure against differential SCA is randomizing the intermediate data, thereby rendering the calculation of the hypothetical leakage values rather impossible. Coron [40] suggested three countermeasures to protect against differential SCA attacks:

1. Blinding the private scalar by adding a multiple of $\#E$. For any random number r and $k' = k + r\#E$, we have $k' \cdot P = k \cdot P$ since $(r\#E) \cdot P = \mathcal{O}$.

2. Blinding the point P, such that $k \cdot P$ becomes $k \cdot (P + R)$. The known value $S = k \cdot R$

is subtracted at the end of the computation.

3. Randomizing the homogeneous projective coordinates (X, Y, Z) with a random $\lambda \neq 0$ to $(\lambda X, \lambda Y, \lambda Z)$. The random variable λ can be updated in every execution or after each doubling or addition.

Very similar, Joye and Tymen [84] suggested to make use of an elliptic curve isomorphism of the fixed curve or of an isomorphic representation of the field. Ciet and Joye [38] also suggested several similar randomization methods:

1. Random key splitting: $k = k_1 + k_2$ or $k = \lfloor k/r \rfloor r + (k \bmod r)$ for a random r.
2. Randomized EC isomorphism.
3. Randomized field isomorphism. We refer to the corresponding paper for a detailed explanation [84].

Coron's first two defense strategies were scrutinized in [114] and judged weak if implemented as presented. The latter three countermeasures are broken by an RPA attack in [56]. (See Section 6.4).

6.4 Comparative Side-Channel Attacks

Comparative SCA resides between a simple SCA and a differential SCA. Two portions of the same or different leakage trace are compared to discover the reuse of values. The umbrella term was introduced in [77], but the first reported attack belonging to this category is the doubling attack. The doubling attack [50] on ECC is an attack with chosen inputs and has been shown powerful to attack some classic SPA-protected algorithms such as left-to-right (downward) double-and-add-always algorithm. The attacker does not need to tell whether a computation being performed is a point doubling or addition. More precisely, for two point

doublings $(2 \times t_1)P$ and $(2 \times t_2)P$, even if the attacker cannot tell the exact values of t_1 or t_2, the attacker can still detect if $t_1 = t_2$.

To thwart this attack, blinding techniques can be effective. Care has to be taken however that neither blinding the base point or the scalar is applied solely. This has been proven insecure [50]. Combined use strengthens the security.

Refined Power Analysis

A refined side-channel analysis attack (RPA is short for Refined Power Analysis) directs its attention to the existence of a point P_0 on the elliptic curve $E(\mathbb{K})$ such that one of the coordinates is 0 in $\mathbb{K}$ and $P_0 \neq \mathcal{O}$. Randomized projective coordinates, randomized EC isomorphisms and randomized field isomorphisms preserve this specific property of the point P_0. Feeding to a device a point P that leads to a special point $R(0, y)$ (or $R(x, 0)$) at step i under the assumption of some specific key bits will generate exploitable side-channel leakage [56, 103].

The attack can be thwarted by using either a cofactor variant of a protocol for points of "small order" or by using isogenous curves for points of "large order". The zero-value point attack (ZPA) generalizes this attack [5]: zero value points in intermediate results are also considered.

Carry-based Attack

The carry-based attack [49], reported by Fouque et al., does not attack the scalar multiplication itself but its countermeasures.

It relies on the carry propagation occurring when long-integer additions are performed as repeated sub-word additions. For instance, on an 8-bit processor, Coron's first countermeasure, $k' = k + r'$ where $r' = r\#E$, is normally performed with repeated 8-bit additions. Let k_i and r'_i denote the i^{th} sub-word of k and r', respectively. Note that k_i is fixed and r'_i is

random in different executions. The crucial observation here is that, when adding k_i with r'_i, the probability of the carry out $c = 1$ depends solely on the value of k_i (the carry-in has negligible impact [49]). The adversary can then monitor the outgoing carry bit of the adder to estimate the probability of $c = 1$. With this probability, the value of k_i can be guessed with high confidence.

So far, no countermeasures have been proposed to thwart this attack.

EM Attacks

Most simple/differential analysis attacks and countermeasures summed up so far are based on power consumption leakage. Most often, electromagnetic radiation is considered as an extension of the power consumption leakage and the attacks/countermeasures are applied without change [110]. While this approach makes sense in most cases, electromagnetic radiation measurements can be made locally [49] and as such circumvent some countermeasures. Specifically crafted attacks or countermeasures for electromagnetic analysis have not been published.

6.5 Fault (Active) Attacks and Countermeasures

Besides passive SCA, adversaries can actively disturb the cryptographic devices and use the erroneous output (or not even the output, but the reaction of the disturbed device) to derive the secret. In order to do so, the adversary needs to induce faults on the victim device. Various methods can be used, such as changing one memory bit with laser or violating setup time with glitches in clock. The difficulty in inducing a fault depends on its precision, both in time as well as in location. Random faults change an operation or a variable at some point during the execution of a cryptographic algorithm. Precise faults change a specific bit of a specific variable at a specific instance during the execution. Clearly, random faults are easier to introduce, and they are less costly, than precise faults.

In this section, we focus on fault attacks and countermeasures on ECSM. General tampering techniques and tamper-resistance methods will be briefly mentioned. Readers who are interested in these methods are referred to [95].

We divide fault attacks into three categories, namely, safe-error based analysis, weak-curve based analysis and differential fault analysis. Safe-error attacks are based on the observation that some errors will not change the results. Weak curve attacks try to move a scalar multiplication from a strong curve to a *weak* curve. The differential fault attacks analyzes the difference between the correct output and erroneous output to retrieve the scalar bit-by-bit.

Safe-error analysis (M-type and C-type)

The concept of safe-error was introduced by Yen and Joye in [85, 141]. Two types of safe-error are reported: C safe-error and M safe-error. What makes safe-error analysis special is that the adversary is not interested in the erroneous results, but simply the fact that the output is affected or not.

C safe-error

The C safe-error makes use of dummy operations that are introduced to achieve SPA resistance. Taking the add-and-double-always algorithms (Alg. 2) as an example, the dummy addition in step 3 makes safe-error possible. The adversary can induce temporary faults in the ALU or memory during the dummy point addition. If the key bit, k_i, is 1, then final results will be faulty. Otherwise, the final results are not affected. The adversary can thus discover one key bit in one execution.

In order to thwart C safe-error analysis, dummy operations should be avoided. For example, instead of double-and-add-always algorithm, Montgomery's powering ladder should be used. If for certain reasons dummy operations can not be avoided, the key stream should

be represented randomly in each point multiplication.

M safe-error

While the C safe-error attack explores the weakness of an algorithm, the M safe-error attack explores the possible safe-error in an implementation. The attack was first proposed by Yen and Joye [141] to attack RSA. However, it also applies to ECSM.

The basic observation of an M safe-error is that faults in some memory blocks will be cleared. Consider Alg. 3 as an example. We assume that a fault is inducted to y of $R[1]$ right after the calculation of λ during the point doubling in step 3. If $k_i = 1$, then the faults on y will be cleared. Otherwise, it propagates to the end of the ECSM. By simply checking whether the result is affected or not, the adversary can reveal k_i.

Joye and Yen [85] proposed a method to prevent this attack. The idea is to eliminate the possibility of inserting safe-errors. Using the modified Montgomery powering ladder [85], any fault in R[0] or R[1] will be detected regardless of the value of k_i.

Weak curve based analysis

In 2000, Biehl et al. [25] described a new type of fault attack on elliptic curve scalar multiplication. They observed that a_6 was not used in a point multiplication.

An implementation of this algorithm for curve E generates correct results for any curve E' that differs from E only in a_6:

$$E' : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a'_6. \tag{6.1}$$

Thus, the adversary can cheat an ECC processor with a point $P' \in E'(\mathbb{F})$ where E' is a cryptographically *weak* curve.

If an ECC processor does not check whether the base point, P, is a valid point on the

specified curve E or not, the adversary can then choose a point $P' \in E'(\mathbb{K})$ and get the result of the scalar multiplication, $Q' = k \cdot P'$. The adversary can then solve DLP in a subgroup of order $r_{P'}$ (the order of P') to retrieve $k_r = k \bmod r_{P'}$. This process can be repeated to generate k_r for different r. At the end, the Chinese Remainder Theorem can be used to retrieve k. This attack also shows us that not all the fault-based attacks require expensive equipments or sophisticated tampering techniques, and that a naive implementation can be broken with almost negligible cost.

The method of *moving* a scalar multiplication from a strong curve E to a weak curve E' was then extended. With the help of faults, the adversary makes use of invalid points [25], invalid curves [39] and twist curves [48] to hit a weak curve. These methods are described below.

Invalid point attacks

The idea of the invalid point attack is to let the scalar multiplication start with a point P' of a weak curve.

If the ECSM is performed without checking the validity of the base point, then no faults need to be inducted. If the ECC processor checks the validity of the base point, the adversary will try to change the point P right after the point validity check. Note this attack requires fault induction at a specific time, thus is much more difficult than the one described above.

For some applications such as EC El-Gamal or ECDSA, y_2 is not present on the output. In this case, the adversary needs to derive $\{E',\ P'(x'_1, y_1), Q'(x'_2, y_2)\}$ from $\{E,\ P(x_1, y_1),\ x'_2\}$. Though it looks difficult, the adversary still has a non-negligible probability to succeed. Readers who are interested can find the complete method in [25].

A possible countermeasure, as suggested in [25, 39], is Point Validation (PV) before and after scalar multiplication. PV checks if a point lies on an elliptic curve or not. If the base point or result does not belong to the original curve, no output should be given.

Invalid curve attacks

Ciet and Joye [39] refined the attack in [25] by loosening the requirements on fault injection. They show that any *unknown* faults, including permanent faults in non-volatile memory or transient faults caused on the bus, in *any* curve parameters, including field representation and curve parameters a_1, a_2, a_3, a_4, may cause information leakage on the scalar k.

Ciet and Joye suggested using error checking codes to ensure the integrity of curve parameters before scalar multiplication.

Twist curve based FA

In 2008, Fouque et al. [48] discovered a new way to hit a possibly *weak* curve, the quadratic twist curve. They observed that a point multiplication routine for some curve E, without using the y-coordinate, gives correct results for ECSM on its twist curve $\tilde{E}$. They also noticed that the twist curves of many cryptographically strong curves are cryptographically weak (see [48] for details). Eq.6.2 defines the twist curve of E, where ε is a quadratic non-residue in $\mathbb{F}_p$.

$$\tilde{E} : (\varepsilon)y^2 = x^3 + ax + b \tag{6.2}$$

For elliptic curves defined over $\mathbb{F}_p$, a random $x \in \mathbb{F}_p$ corresponds to a point on either E or its twist. Since the order of E and $\tilde{E}$ are close, the probability is approximatively one half that a random abscissa corresponds to a point on E or $\tilde{E}$. As a result, the adversary has a probability of one half to hit a point on $\tilde{E}$ with a random fault on x-coordinate of P on E.

There are three possible methods to thwart this attack. The first one is to repeat point validity check during the scalar multiplication. The second one is to use y-coordinate all the time. Both methods have some overhead in terms of computation time and storage. The third one is to choose twist-secure curves, namely, curves whose twist curve are also cryptographically strong.

Differential FA

The DFA attacks uses the difference between the correct results and the faulty results to deduce certain bits of the scalar.

1) Biehl-Meyer-Müller DFA

Biehl et al. [25] reported the first DFA on an ECSM. We use an right-to-left multiplication algorithm to describe this attack. Let Q_i and R_i denote the value of Q and R at the end of the i^{th} iteration, respectively. Let $k(i) = k$ div 2^i. Let Q'_i be the value of Q if faults have been induced. The attack reveals k from the Most Significant Bits (MSB) to the Least Significant Bits (LSB).

1. Run ECSM once and collect the correct result (Q_n).
2. Run the ECSM again and induce an one-bit flip on Q_i, where $l - m \leq i < l$. We assume that m is *small*.
3. Note that Q_n=Q_i+$(k(i)2^i)P$ and Q'_n=Q'_i+$(k(i)2^i)P$. The adversary then tries all possible $k(i) \in \{0, 1, .., 2^m - 1\}$ to generate Q_i and Q'_i. The correct value of $k(i)$ will result in a $\{Q_i$,$Q'_i\}$ that have only one-bit difference.

The attack works for left-to-right multiplication algorithm as well. It also applies if k is encoded with any other deterministic codes such as Non-Adjacent-Form (NAF) and w-NAF.

Algorithm 4 Right-To-Left (upwards) binary method for point multiplication

Input: $P \in E(\mathbb{F})$ and integer $k = \sum_{i=0}^{l-1} k_i 2^i$.
Output: kP.
1: $R \leftarrow P$, $Q \leftarrow \mathcal{O}$.
2: **for** $i = 0$ to $l - 1$ **do**
3: If $k_i = 1$ then $Q \leftarrow Q + R$.
4: $R \leftarrow 2R$.
5: **end for**
Return R.

It is also claimed that a fault induced at random moments during an ECSM is sufficient [25].

To thwart this attack, the validity of the intermediate results (Q_i and R_i in Algorithm 4) should be regularly checked. Another possible countermeasure is to randomize the scalar k such that the adversary can does not gain more bits of k in repeated executions.

2) Sign change FA

In 2006, Blömer et al. [27] proposed the sign change fault (SCF) attack. It attacks implementations where scalar is encoded in Non-Adjacent Form (NAF). When using curves defined over the prime field, the sign change of a point implies only a sign change of its y-coordinate. The SCF attack does not force the elliptic curve operations to leave the original group $E(\mathbb{F}_p)$, thus P is always a valid point.

A straightforward countermeasure against an SCF attack is to use Montgomery ladder algorithm that does not use the y-coordinate for computing ECSM (e.g. Montgomery Scalar Multiplication with López-Dahab coordinates [104]). Another countermeasure presented by Blömer et al. [27] uses a second elliptic curve whose order is a small prime number to verify the final results.

6.6 Guidelines to Select Countermeasures

One can not simply integrate all the countermeasures discussed above to thwart all attacks. The reasons for this are manifold. The complexity and extra overhead added by countermeasures can significantly increase the design and manufacturing cost. Another important reason is that a countermeasure against one attack may benefit another one. Thus, countermeasures should be carefully selected such that they do not add extra vulnerabilities. In this section, we discuss the cross relationship between known attacks and countermeasures.

Countermeasures *vs.* **Attacks**

Table 6.1 summarizes the most important attacks and their countermeasures. The different attacks, grouped into passive attacks and active attacks are listed column-wise, while each row represents one specific countermeasure. Let A_j and C_i denote the attack in the j^{th} column and countermeasure in the i^{th} row, respectively. The grid (i, j), the cross of the i^{th} row and the j^{th} column, shows the relation between A_j and C_i.

- $\surd$: C_i is an effective countermeasure against A_j.
- $\times$: C_i is attacked by A_j.
- **H**: C_i helps A_j.
- **?**: C_i might be an effective countermeasure against A_j, but the relation between C_i and A_j is unclear or unpublished.
- $-$: C_i and A_j are irrelevant (C_i is not effective against A_j).

It is important to make a difference between $\times$ and $-$. Here $\times$ means C_i is attacked by A_j, where $-$ means that the use of C_i does not affect the effort or result of A_j at all. For example, scalar randomization using 20-bit random number is attacked by doubling attack, so we put a $\times$ at their cross. The Montgomery powering ladder is designed to thwart SPA, and it does not make a DPA attack harder or easier, so we put a $-$ there.

Below we discuss each countermeasure and its relation with the listed attacks.

Indistinguishable Point Addition Formulae. Indistinguishable group operations render a simple SCA impossible, but only if the underlying field arithmetic is implemented securely. This is discussed in [127, 135]. This method does not counteract differential SCA and RPA/ZPA [81].

Table 6.1: Attacks versus Countermeasures

	Passive Attacks							Active Attacks						
								Safe-Error		Weak Curve			Differential	
	TA	SPA/SEMA	Template Attack	DPA/DEMA	Comparative SCA	RPA/ZPA	Carry-based Attack	M Safe-Error	C Safe-Error	Invalid Point	Invalid Curve	Twist Curve	Sign Change	Differential
Indistinguishable Point Addition [30]	√	√	–	–	**?**	–	–	–	–	–	–	–	–	–
Double-and-add-always [40]	√	√	–	–	× [50]	–	–	–	×**H** [141]	–	–	–	–	–
Montgomery Powering Ladder⊢ [85]	√	√	–	–	× [143]	× [5]	–	√	√	–	–	**H** [48]	√	–
Montgomery Powering Ladder⊣ [85]	√	√	–	–	× [143]	× [5]	–	√	√	–	–	√	–	–
Random scalar split [38]	–	–	**?**	√	**?**	√	×	–	**?**	–	–	√	**?**	**?**
Scalar randomization [40]	–	–	× [107]	× [114]	× [50]	√	× [49]	–	**?**	–	–	–	**?**	**?**
Base point blinding [40]	–	–	× [107]	× [114]	× [50]	√	–	–	–	**?**	–	–	–	**?**
Random Projective Coordinates [40]	–	–	√	√	**?**	× [56]	–	–	–	–	–	–	–	**?**
Randomized EC Isomorphisms [38]	–	–	**?**	√	**?**	× [56]	–	–	–	–	–	–	–	**?**
Randomized Field Isomorphisms [38]	–	–	**?**	√	**?**	× [56]	–	–	–	–	–	–	–	**?**
Point validity check [25]	–	–	–	–	–	–	–	–	**H**	√	**?**	√	×**H** [?]	√
Curve integrity check [39]	–	–	–	–	–	–	–	–	–	**?**	√	–	–	–
Coherence check [41]	–	–	–	–	–	–	–	–	**H**	–	**?**	–	√	√

Let A_j and C_i denote the attack in the j^{th} column and countermeasure in the i^{th} row, respectively.

- √: C_i is an effective countermeasure against A_j.
- ×: C_i is attacked by A_j.
- **H**: C_i helps A_j.
- **?**: C_i might be an effective countermeasure against A_j, but the relation between C_i and A_j is unclear or unpublished.
- –: C_i and A_j are irrelevant (C_i is not effective against A_j).
  ⊣: using y-coordinate
  ⊢: without using y-coordinate

Double-and-add-always. The double-and-add-always algorithm is the main representative of the countermeasures that use dummy instructions or operations to withstand simple side-channel attacks.

The algorithm fails against doubling attacks. It does not remove vulnerabilities to differential SCA attacks. It also makes C safe-error fault attack possible.

Montgomery Powering Ladder. The Montgomery powering ladder is an algorithm-level countermeasure running in a fixed time without redundant operations, hence it is SCA resistant. It avoids the usage of dummy instructions and also resists the *normal* doubling attack. However, it is attacked by the relative doubling attack proposed by Yen et al. [143]. This attack can reveal the relation between two adjacent secret scalar bits, thereby seriously decreases the number of key candidates.

With Montgomery powering ladder, y-coordinate is not necessary during the scalar multiplication, which prevents sign-change attacks. However, for curves that have weak twist curves, using Montgomery powering ladder without y-coordinate is vulnerable to twist curve attacks.

Random scalar split. This countermeasure can resist DPA/DEMA attacks since it has a random scalar for each execution. In [50], the authors have already analyzed the effectiveness of Coron's first countermeasure against the doubling attack. If we assume that the scalar k is randomly split into two full length scalars, the search space is extended to 2^{81} for a 163-bit k (the birthday paradox applies here). This is enough to resist the doubling attack. It can also help to thwart RPA/ZPA if it is used together with base point randomization [5, 56, 70].

However, this countermeasure is vulnerable to a carry-based attack if the key is split as follows: choosing a random number $r < \#E$, and $k_1 = r$, $k_2 = k - r$.

Scalar randomization. With respect to the resistance against passive SCA, the above analysis of the random scalar split countermeasure against DPA/DEMA and RPA/ZPA also

applies here. However, as mentioned in [50] the 20-bit random value for blinding the scalar k is not enough to resist the doubling attack.

Like random scalar split, it renders the safe-error and sign-change attacks more difficult. On the other hand, it is shown in [49] that the key randomization process, namely, $k' = k + r\#E$, leaks the scalar under the carry-based attack.

Base point blinding. For an ECSM, the scalar randomization and base point blinding are based on the same idea of randomizing one component of the point multiplication. Therefore, their effectiveness against various passive attacks is similar. It can resist DPA/DEMA as explained in [40]. In [50], the authors conclude that this countermeasure is still vulnerable to the doubling attack since the point which blinds P is also doubled at each execution. This countermeasure makes RPA/ZPA more difficult since it can break the assumption that the attacker can freely choose the base point (the base point is blinded).

This countermeasure might make the weak-curve based attacks more difficult since the attacker does not know the masking point R. In an attack based on an invalid point, the adversary needs to find out the faulty points P' and $Q' = k \cdot P'$. With the point blinding, it seams to be more difficult to reveal either P' or Q'. However, in the case of an invalid curve attack, base point blinding does not make a difference.

Random projective coordinates. This countermeasure is effective against differential SCA. It fails to resist the RPA as zero is not effectively randomized. Combination with a simple SCA countermeasure is essential.

Point validity check. This countermeasure checks if a certain point is on the authentic curve or not. It is an effective countermeasure against invalid point attacks. If the y-coordinate is used, it is also effective against a twist-curve attack. However, it is not effective against invalid curve attacks, sign-change attacks and C safe-error attacks.

Curve integrity check. The curve integrity check is to detect fault injections on curve parameters. Before starting an ECSM the curve parameters will be read from the non-volatile

memory (possibly on the data bus), which are vulnerable to permanent or transient faults. So, the integrity of the curve parameters (including the base point) needs to be verified using a CRC (cyclic redundancy check) code before an ECSM execution.

Coherence check. A coherence check verifies the intermediate or final results with respect to a valid pattern. If an ECSM uses the Montgomery powering ladder, we can use the fact that the difference between $R[0]$ and $R[1]$ is always P. This can be used to detect faults during an ECSM [41].

Selecting Countermeasures

After analyzing the existing attacks and countermeasures, a natural question is whether there exists a set of countermeasures that resists all the existing passive and active attacks. While unified countermeasures to tackle both the passive and active attacks are attractive, they are very likely weaker than what is expected. Baek and Vasyltsov extended Shamir's trick, which was proposed for RSA-CRT, to secure ECC from DPA and FA [13]. However, Joye showed in [83] that a non-negligible portion of faults was undetected using the unified countermeasure and settings in [13]. In this section, we describe several principles to choose countermeasures.

Complete: A complete picture of attacks and countermeasures is the perfect base to select countermeasures. As we pointed out above, an adversary needs to succeed in only one out of many possible attack methods to win. Keeping a summary of up-to-date attacks and countermeasures is important for cryptosystem designers.

Specific: Whenever selecting countermeasures for a cryptosystem, a detailed description of the cryptosystem should be explicitly defined. A set of countermeasures that can thwart all known attacks is neither easy to find nor efficient in terms of area and performance. Within restricted boundaries, countermeasure selection is much easier and more efficient. For example, RPA and comparative SCA assume that the attacker can choose the base

point freely. If an ECC processor is targeting an application where the base point is fixed, then an RPA and doubling attack can not apply.

Additive: The selected countermeasures should be additive. Suppose that we choose countermeasures from Table 6.1, we could proceed in two steps.

The first step is a column-wise selection. We inspect each column and select a countermeasure that suffices to thwart the attack in this column. If we have chosen two countermeasures, C_a and C_b, and their relation with A_j is as follows: $(a, j) = \surd$, $(b, j) = \times$. In this case, we need to study whether C_a covers C_b or not. **H** in the table should be avoided whenever possible. If eventually we can not get rid of all the **H**, extra countermeasures should be added to cover it.

The second step is to check if the selected countermeasures are additive. Using multiple countermeasures simultaneously might introduce new vulnerabilities. Thus, we need to evaluate the selected countermeasures as a new countermeasure.

6.7 Conclusion

Since the whole AES and PRESENT are implemented in hardware with the current HW/SW partitioning, many existing SCA attack countermeasures can be applied to protect the AES/PRESENT coprocessor [79]. However, when we re-examine the security of the SoC after integration of these two block ciphers, it is easy to find that the secret key that will be transmitted from microprocessor to the AES and PRESENT coprocessors through the bus can be easily compromised by probing the bus. Not only the key distribution may impose additional security holes, but also the key generation and storage may also be attacked. Therefore, the security of cryptographic primitives should be re-evaluated after system integration as a cryptosystem will fail at its weakest link. For this purpose, we use ECC as a case study and perform a comprehensive threat analysis in the ECC cryptosystem design. We strongly believe that keeping track of the ever evolving field of implementation attacks is

a titanic work for a cryptosystem designer. Our research provides a digest of existing attacks and countermeasures.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This dissertation aims to provide a systematic way to design secure and efficient cryptographic primitives. After identifying the main research problems in Chapter 1 and introducing the background in Chapter 2, we started with how to use HW/SW codesign techniques to approach an ideal coprocessor architecture, which shows a good tradeoff between performance, cost, security and flexibility in Chapter 3. A methodology for comprehensive performance evaluation of NIST SHA-3 competition candidates hardware implementations was presented in Chapter 4. As a further discussion in how to do fair comparison of cryptographic hardware, in Chapter 5 the impact of technology to the comparison of SHA-3 FPGA and ASIC benchmarking results and cost analysis of lightweight hash implementations are presented. Finally, in Chapter 6 we have presented a survey of existing implementations attacks and countermeasures on an ECC cryptosystem and provided guidelines to systematically select countermeasures to protect it from multiple types of attacks.

The main contributions were discussed in Chapter 3, 4, 5, and 6.

Main conclusion for Chapter 3. *The hardware profile of a standalone cryptographic*

coprocessor is not a good measure for the performance and energy/power in the context of a real embedded system. The HW/SW partitioning and HW/SW interface selection will play a more important role in the design efficiency since HW/SW communication will also become an integral part of the cryptographic computation process. A tradeoff between cost, performance, flexibility and security needs to be considered in the HW/SW codesign of a cryptographic coprocessor.

Main conclusion for Chapter 4. *The hardware benchmarking of different cryptographic algorithms based on a fair and comprehensive methodology can be very challenging because of the undefined application scenarios, various choices of technologies and multiple optimization goals. In the context of NIST SHA-3 competition, we described our efforts in ASIC performance evaluation. We first described the overall design flow that combines FPGA prototyping with ASIC design, and next elaborated the efforts to automate and standardize the ASIC implementation process, and finally presented how to design and benchmark a SHA-3 ASIC.*

Main conclusion for Chapter 5. *There are two technology-dependent issues in the evaluation of cryptographic hardware: how to perform cross-platform comparison between the FPGA and ASIC results and how to conduct technology-dependent cost analysis. In the context of hash function designs, we provided some insights on how to look at SHA-3 hardware benchmarking results in different platforms. Our technology-dependent cost analysis may help cryptographic engineers have better presentation of the metrics and avoid some common pitfalls.*

Main conclusion for Chapter 6. *A cryptosystem will fail at its weakest link and one countermeasure against one SCA or fault attack may benefit another attack. Keeping track of the ever evolving field of implementation attacks is a titanic work for a cryptosystem designer, and this chapter provides a digest of existing attacks and countermeasures. Three guidelines were proposed and explained in the selection of countermeasures: complete, specific and additive.*

7.2 Future Work

As new cryptographic standards come out from time to time, and the implementation efficiency and hardware security need to satisfy different requirements of many novel security applications. The research presented in this dissertation can serve as the base for several future research directions.

In Chapter 4 we defined a SHA-3 ASIC evaluation methodology and presented hardware evaluation results for the SHA-3 Fourteen Second Round and Five Third Round candidates. However, the final SHA-3 ASIC benchmarking results only reflected the chosen fixed ASIC library, technology node, synthesis constraints, and floor plan shape. One future research direction toward more comprehensive ASIC benchmarking process is to explore more ASIC technology impact factors as mentioned above. The SASEBO-R board used as the SHA-3 ASIC testing platform and the SASEBO-GII board used as the SHA-3 FPGA prototyping were both originally developed for side-channel analysis. Hence, a potential research area is side-channel analysis of SHA-3 candidates when implemented in FPGAs and ASICs.

In Chapter 5 we proposed a lightweight hash design methodology and optimized the storage elements with much more compact register file. We also used one of the SHA-3 Second Round candidates, CubeHash, as a case study to show the effectiveness of our proposed method. Future work in this direction may include the discussion of applying the lightweight hash design methodology to the optimization of all the five SHA-3 finalists. It is also interesting to see whether similar optimizations in storage structure can be found in lightweight block cipher designs.

In Chapter 6 we conducted a survey on implementation attacks and countermeasures on ECC with main results summarized in Table 6.1. Though physical security of cryptographic hardware or software has been intensively studied in the last ten years, known methods to protect physical attacks are far from satisfactory. For future research, we believe the following topics are important to improve our understanding in physical security of cryptosystems.

First, mathematical model to evaluate the effectiveness of attacks and countermeasures. For example, an attack requires certain amount of information leakage to reveal the scalar, which sets up an upper bound of information leakage for an effective countermeasure. Models that allow a quantitative evaluation of physical information leakage are still missing. Second, a framework to evaluate the effectiveness of a set of countermeasures. Multiple countermeasures are always used together to thwart multiple attacks. However, the current method for choosing countermeasures is rather ad-hoc. Third, Table 6.1 shows that we only understand a small part of the complete picture, and many interesting attack-countermeasure pairs have not been studied yet. Finally, system integration of multiple countermeasures. In [59], we suggested a combined countermeasure and discussed the system integration cost. A perfect countermeasure is probably useless if it is too complex to implement.

Publication List

Journals, Conference Proceedings and Reports:

[1] M. Srivistav, X. Guo, S. Huang, D. Ganta, M. B. Henry, L. Nazhandali, and P. Schaumont. Design and Benchmarking of an ASIC with Five SHA-3 Finalist Candidates. *Microprocessors and Microsystems: Embedded Hardware Design (MICPRO)*. (*to appear*)

[2] X. Guo, M. Srivistav, S. Huang, D. Ganta, M. B. Henry, L. Nazhandali, and P. Schaumont. ASIC Implementations of Five SHA-3 Finalists. *Design, Automation and Test in Europe (DATE2012)*, pp.1006-1011, Mar. 2012.

[3] X. Guo and P. Schaumont. The Technology Dependence of Lightweight Hash Implementation Cost. *ECRYPT Workshop on Lightweight Cryptography (LC2011)*, Nov. 2011.

[4] X. Guo, M. Srivistav, S. Huang, D. Ganta, M. B. Henry, L. Nazhandali, and P. Schaumont. Pre-silicon Characterization of NIST SHA-3 Final Round Candidates. *14th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2011)*, pp.535-542, Aug. 2011.

[5] X. Guo, Meeta Srivistav, S. Huang, L. Nazhandali, and P. Schaumont. VLSI Characterization of NIST SHA-3 Finalists. *48th Design Automation Conference (DAC 2011) Work-In-Progress (WIP)*, Jun. 2011.

[6] X. Guo, M. Srivistav, S. Huang, D. Ganta, M. B. Henry, L. Nazhandali, and P. Schaumont. Silicon Implementation of SHA-3 Finalists: BLAKE, Grøstl, JH, Keccak and

Skein. *ECRYPT II Hash Workshop 2011*, May 2011.

[7] Z. Chen, X. Guo, A. Sinha, and P. Schaumont. Data-Oriented Performance Analysis of SHA-3 Candidates on FPGA Accelerated Computers. *Design, Automation and Test in Europe (DATE2011)*, pp.1-6, 2011.

[8] X. Guo and P. Schaumont. Optimized System-on-Chip Integration of a Programmable ECC Coprocessor. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol.4, no.1, pp.6:1-6:21, Dec. 2010.

[9] X. Guo, S. Huang, L. Nazhandali, and P. Schaumont. On The Impact of Target Technology in SHA-3 Hardware Benchmark Rankings. *Cryptology ePrint Archive, Report 2010/536*, 2010.

[10] X. Guo, S. Huang, L. Nazhandali, and P. Schaumont. Fair and Comprehensive Performance Evaluation of 14 Second Round SHA-3 ASIC Implementations. *NIST 2nd SHA-3 Candidate Conference*, Aug. 2010.

[11] K. Kobayashi, J. Ikegami, M. Knezevid, X. Guo, S. Matsuo, S. Huang, L. Nazhandali, U. Kocabas, J. Fan, A. Satoh, I. Verbauwhede, K. Sakiyama, and K. Ota. A Prototyping Platform for Performance Evaluation of SHA-3 Candidates. *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST2010)*, pp.60-63, Jun. 2010.

[12] J. Fan, X. Guo, E. DeMulder, P. Schaumont, and I. Verbauwhede. State-of-the-art of Secure ECC Implementations: A Survey on Known Side-Channel Attacks and Countermeasures. *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST2010)*, pp.76-87, Jun. 2010.

[13] X. Guo, J. Fan, P. Schaumont, and I. Verbauwhede. Programmable and Parallel ECC Coprocessor Architecture: Tradeoffs between Area, Speed and Security. *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2009)*, LNCS5747, pp.289-303, Sep. 2009.

[14] X. Guo and P. Schaumont. Optimizing the Control Hierarchy of an ECC Copro-

cessor Design on an FPGA based SoC Platform. *5th International Workshop on Applied Reconfigurable Computing (ARC2009)*, LNCS5453, pp.169-180, Springer Verlag, Feb. 2009.

[15] X. Guo and P. Schaumont. Optimizing the HW/SW Boundary of an ECC SoC Design Using Control Hierarchy and Distributed Storage. *Design, Automation and Test in Europe (DATE2009)*, pp.454-459, Apr. 2009.

[16] Z. Chen, X. Guo, R. Nagesh, A. Reddy, M. Gora, and A. Maiti. Hardware Trojan Designs on BASYS FPGA Board. *Embedded System Challenge Contest in Cyber Security Awareness Week (CSAW08)*, 2008.

[17] X. Guo, Z. Chen, and P. Schaumont. Energy and Performance Evaluation of an FPGA-based SoC Platform with AES and PRESENT Coprocessors. *International Workshop on Systems, Architectures, Modeling, and Simulation (SAMOS 2008)*, LNCS5114, pp.106-115, Springer Verlag, Jul. 2008.

[18] J. Xing, X. Zou, and X. Guo. Ultra-Low Power S-Boxes Architecture for AES. *The Journal of China University of Posts and Telecommunications*, Vol.15, no.1, pp.112-117, Mar. 2008.

[19] Z. Liu, J. Xiao, X. Zou, and X. Guo. Edge-based Algorithm of Real-time Image Resizing. *Journal of Image and Graphics*, vol.13, no.2, pp.225-229, 2008. (*in Chinese*)

[20] J. Xiao, X. Zou, Z. Liu, and X. Guo. A Novel Adaptive Interpolation Algorithm for Image Resizing. *International Journal of Innovative Computing, Information and Control*, vol.3, no.6(A), pp.1335-1345, 2007.

[21] J. Zhang, Z. Liu, X. Zou, and X. Guo. Design of Timing Controller for LCD System. *Computer and Digital Engineering*, vol.35, no.3, pp.151-154, 2007. (*in Chinese*)

[22] J. Xiao, X. Zou, Z. Liu and X. Guo. The Research of an Adaptive Algorithm for Real-time Image Enhancement. *Mircoelectronics & Computer*, vol.23, no.5, pp.15-17, 2006. (*in Chinese*)

[23] X. Guo, Z. Liu, J. Xing, W. Fan and X. Zou. Optimized AES Crypto Design for Wireless Sensor Networks with a Balanced S-box Architecture. *International Conference on Informatics and Control Technologies (ICT2006)*, pp.203-208, IET, 2006.

[24] Z. Liu, X. Guo, X. Zou, and J. Xiao. Image Color Enhancement Technique Based on Improved Bayer Dithering Algorithm. *Journal of Huazhong Univ. of Science & Technology (Nature Science)*, vol.34, no.5, pp.68-70, 2006. (*in Chinese*)

[25] Z. Liu, X. Guo, Y. Chen, Y. Han and X. Zou. On the Ability of AES SBoxes to Secure Against Correlation Power Analysis. *3rd Information Security Practice and Experience Conference (ISPEC 2007)*, LNCS 4464, pp.43-50, Springer Verlag, 2007.

[26] J. Xiao, X. Zou, Z. Liu, X. Guo. Adaptive Interpolation Algorithm for Real-time Image Resizing. *International Conference on Innovative Computing, Information and Control (ICICIC'06)*, vol.2, pp.221-224, IEEE, 2006.

[27] X. Guo, Z. Liu, X. Zou, J. Xiao and H. Zhao. Picture Sharpness Module Design for Scaler. *Computer and Digital Engineering*, vol.33, no.5, pp.82-84, 2005. (*in Chinese*)

Bibliography

[1] D. Agrawal, B. Archambeault, J. Rao, and P. Rohatgi. The EM Side-Channel(s). In Burton Kaliski, Çetin Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer Berlin / Heidelberg, 2003.

[2] H. Aigner, H. Bock, M. Htter, and J. Wolkerstorfer. A Low-Cost ECC Coprocessor for Smartcards. In *Proceedings of Cryptographic Hardware and Embedded Systems*, pages 107–118, Berlin Heidelberg, January 2004. Springer.

[3] AIST-RCIS. SHA-3 Hardware Project, May 2011. `http://www.rcis.aist.go.jp/special/SASEBO/SHA3-en.html`.

[4] AIST-RCIS. Side-Channel Attack Standard Evaluation Board, May 2011. `http://staff.aist.go.jp/akashi.satoh/SASEBO/en/index.html`.

[5] T. Akishita and T. Takagi. Zero-Value Point Attacks on Elliptic Curve Cryptosystem. In Colin Boyd and Wenbo Mao, editors, *Information Security*, volume 2851 of *Lecture Notes in Computer Science*, pages 218–233. Springer Berlin / Heidelberg, 2003.

[6] H. Alrimeih and D. Rakhmatov. Security-Performance Trade-offs in Embedded Systems Using Flexible ECC Hardware. *IEEE Design & Test*, 24(6):556–569, 2007.

[7] F. Amiel, K. Villegas, B. Feix, and L. Marcel. Passive and Active Combined Attacks: Combining Fault Attacks and Side Channel Analysis. In *Proceedings of the Workshop*

on Fault Diagnosis and Tolerance in Cryptography, pages 92–102, Washington, DC, USA, 2007. IEEE Computer Society.

[8] J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia. Quark: A Lightweight Hash. In Stefan Mangard and Francois-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *LNCS*, pages 1–15. 2010.

[9] J.-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia. Quark: a lightweight hash, October 2011. `http://131002.net/quark/`.

[10] R. Avanzi. Side Channel Attacks on Implementations of Curve-Based Cryptographic Primitives. Cryptology ePrint Archive, Report 2005/017. Available from http://eprint.iacr.org/.

[11] R. M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. CRC Press, 2005.

[12] S. Badel, N. Dağtekin, J. Nakahara, K. Ouafi, N. Reffé, P. Sepehrdad, P. Sušil, and S. Vaudenay. ARMADILLO: A Multi-purpose Cryptographic Primitive Dedicated to Hardware. In Stefan Mangard and Francois-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 398–412. Springer Berlin / Heidelberg, 2010.

[13] Y.-J. Baek and I. Vasyltsov. How to Prevent DPA and Fault Attack in a Unified Way for ECC Scalar Multiplication - Ring Extension Method. In *Information Security Practice and Experience(ISPEC2007), LNCS 4464*, pages 225–237, Berlin/Heidelberg, 2007. Springer.

[14] B. Baldwin, N. Hanley, M. Hamilton, L. Lu, A. Byrne, M. ONeill, and W. P. Marnane. FPGA Implementations of SHA-3 Candidates: CubeHash, Grøstl, LANE, Shabal and Spectral Hash. Cryptology ePrint Archive, Report 2009/342, 2009. `http://eprint.iacr.org/2009/342`.

[15] 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. In *The Second SHA-3 Candidate Conference*, August 2010.

[16] L. Batina, D. Hwang, A. Hodjat, B. Preneel, and I. Verbauwhede. Hardware/software co-design for hyperelliptic curve cryptography (HECC) on the 8051 μP. In *Proceedings of Cryptographic Hardware and Embedded Systems*, pages 106–118, Berlin Heidelberg, January 2005. Springer.

[17] M. Bernet, L. Henzen, H. Kaeslin, N. Felber, and W. Fichtner. Hardware implementations of the SHA-3 candidates Shabal and CubeHash. *Circuits and Systems, Midwest Symposium on*, pages 515–518, 2009.

[18] D. Bernstein. ChaCha, a variant of Salsa20, January 2008. `http://cr.yp.to/chacha/chacha-20080128.pdf`.

[19] D. Bernstein. CubeHash: 16-cycle-per-round hardware implementation strategy, May 2011. `http://cubehash.cr.yp.to/hardware16/hash.c`.

[20] D. Bernstein. CubeHash: 32-cycle-per-round hardware implementation strategy, May 2011. `http://cubehash.cr.yp.to/hardware32/hash.c`.

[21] D. Bernstein. CubeHash: 8-cycle-per-round hardware implementation strategy, May 2011. `http://cubehash.cr.yp.to/hardware8/hash.c`.

[22] D. Bernstein. CubeHash: a simple hash function, May 2011. `http://cubehash.cr.yp.to/index.html`.

[23] D. Bernstein and T. Lange. eBACS: ECRYPT Benchmarking of Cryptographic Systems, May 2011. `http://bench.cr.yp.to`.

[24] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The Keccak sponge function family – Updated VHDL package, May 2011. `http://keccak.noekeon.org/VHDL_3.0.html`.

[25] I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Elliptic Curve Cryptosystems. In *CRYPTO*, volume 1880, pages 131–146. Springer, 2000.

[26] I. Blake, G. Seroussi, N. Smart, and J. W. S. Cassels. *Advances in Elliptic Curve Cryptography (London Mathematical Society Lecture Note Series)*. Cambridge University Press, New York, NY, USA, 2005.

[27] J. Blömer, M. Otto, and J.-P. Seifert. Sign Change Fault Attacks on Elliptic Curve Cryptosystems. In *Fault Diagnosis and Tolerance in Cryptography(FDTC2006), LNCS 4236*, pages 36–52, Berlin/Heidelberg, 2006. Springer.

[28] A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varici, and I. Verbauwhede. SPONGENT: A Lightweight Hash Function. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems, CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 312–325. Springer Berlin / Heidelberg, 2011.

[29] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M. Robshaw, and Y. Seurin. Hash Functions and RFID Tags: Mind the Gap. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems C CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 283–299. Springer Berlin / Heidelberg, 2008.

[30] E. Brier and M. Joye. WeierstraßElliptic Curves and Side-Channel Attacks. In *PKC '02: Proceedings of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems*, pages 335–345, London, UK, 2002. Springer-Verlag.

[31] CAIDA. Packet size distribution comparison between Internet links in 1998 and 2008, July 2011. `http://www.caida.org/research/traffic-analysis/pkt_size_distribution/graphs.xml`.

[32] C. Cannière, O. Dunkelman, and M. Knežević. KATAN and KTANTAN – A Family of Small and Efficient Hardware-Oriented Block Ciphers. In *Proceedings of the 11th*

International Workshop on Cryptographic Hardware and Embedded Systems, CHES '09, pages 272–288, Berlin, Heidelberg, 2009. Springer-Verlag.

[33] S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In *Cryptographic Hardware and Embedded Systems, CHES 2002*, pages 13–28, 2002.

[34] Z. Chen, S. Morozov, and P. Schaumont. A Hardware Interface for Hashing Algorithms. Cryptology ePrint Archive, Report 2008/529, 2008. `http://eprint.iacr.org/2008/529`.

[35] R. C. C. Cheung, W. Luk, and P. Y. K. Cheung. Reconfigurable Elliptic Curve Cryptosystems on a Chip. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 24–29, Washington, DC, USA, 2005. IEEE Computer Society.

[36] B. Chevallier-Mames, M. Ciet, and M. Joye. Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity. *IEEE Trans. Computers*, 53(6):760–768, 2004.

[37] P. Chodowiec, P. Khuon, and K. Gaj. Fast Implementations of Secret-Key Block Ciphers Using Mixed Inner- and Outer-Round Pipelining. In *FPGA 2001*, pages 94 –102. ACM, 2001.

[38] M. Ciet and M. Joye. (Virtually) Free Randomization Techniques for Elliptic Curve Cryptography. In *Information and Communications Security (ICICS2006), LNCS 2836*, pages 348–359, Berlin/Heidelberg, 2003. Springer.

[39] M. Ciet and M. Joye. Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. *Des. Codes Cryptography*, 36(1):33–43, 2005.

[40] J. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In *CHES '99: Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, pages 292–302, London, UK, 1999. Springer-Verlag.

[41] A. Dominguez-Oviedo. *On Fault-based Attacks and Countermeasures for Elliptic Curve Cryptosystems*. PhD thesis, University of Waterloo, Canada, 2008.

[42] ECRYPT. The eSTREAM project, May 2011. `http://www.ecrypt.eu.org/stream/`.

[43] ECRYPT. The SHA-3 Zoo, May 2011. `http://ehash.iaik.tugraz.at/wiki/SHA-3_Hardware_Implementations`.

[44] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel. A Survey of Lightweight-Cryptography Implementations. *IEEE Des. Test*, 24:522–533, November 2007.

[45] M. Feldhofer and C. Rechberger. A Case Against Currently Used Hash Functions in RFID Protocols. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, volume 4277 of *Lecture Notes in Computer Science*, pages 372–381. Springer Berlin / Heidelberg, 2006.

[46] M. Feldhofer and J. Wolkerstorfer. Strong Crypto for RFID Tags - A Comparison of Low-Power Hardware Implementations. In *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pages 1839 –1842, may 2007.

[47] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES implementation on a grain of sand. *Information Security, IEE Proceedings*, 152(1):13 – 20, oct. 2005.

[48] P. Fouque, R. Lercier, D. Réal, and F. Valette. Fault Attack on Elliptic Curve Montgomery Ladder Implementation. In *Fifth International Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC'08*, pages 92–98. IEEE Computer Society, 2008.

[49] P. Fouque, D. Réal, F. Valette, and M. Drissi. The Carry Leakage on the Randomized Exponent Countermeasure. In *Cryptographic Hardware and Embedded Systems - CHES 2008*, volume 5154, pages 198–213. Springer, 2008.

[50] P.-A. Fouque and F. Valette. The Doubling Attack : Why Upwards Is Better than Downwards. In *Cryptographic Hardware and Embedded Systems - CHES 2003, LNCS2779*, pages 269–280, Berlin/Heidelberg, 2003. Springer.

[51] K. Gaj, E. Homsirikamol, and M. Rogawski. Comprehensive Comparison of Hardware Performance of Fourteen Round 2 SHA-3 Candidates with 512-bit Outputs Using Field Programmable Gate Arrays. NIST 2nd SHA3 Candidates Conference, 2010.

[52] K. Gaj, E. Homsirikamol, and M. Rogawski. Fair and Comprehensive Methodology for Comparing Hardware Performance of Fourteen Round Two SHA-3 Candidates Using FPGAs. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *LNCS*, pages 264–278. Springer, 2010.

[53] Kris Gaj, Jens-Peter Kaps, Venkata Amirineni, Marcin Rogawski, Ekawat Homsirikamol, and Benjamin Y. Brewster. Athena - automated tool for hardware evaluation: Toward fair and comprehensive benchmarking of cryptographic hardware using fpgas. *International Conference on Field Programmable Logic and Applications*, 0:414–421, 2010.

[54] C. Giraud. An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. *IEEE Transactions on Computers*, 55(9):1116–1120, 2006.

[55] T. Good and M. Benaissa. AES from The Fastest to The Smallest. In *Cryptographic Hardware and Embedded Systems, CHES 2005*, volume 3659 of *LNCS*, pages 427 –440. Springer, 2005.

[56] L. Goubin. A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In *PKC '03: Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography*, pages 199–210, London, UK, 2003. Springer-Verlag.

[57] J. Grossschadl. A Low-Power Bit-Serial Multiplier for Finite Fields GF(2^m). In *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on*, volume 4, pages 37 –40 vol. 4, May 2001.

[58] J. Guo, T. Peyrin, and A. Poschmann. The PHOTON Family of Lightweight Hash Functions. In Phillip Rogaway, editor, *Advances in Cryptology, CRYPTO 2011*, volume 6841 of *LNCS*, pages 222–239. Springer Berlin / Heidelberg, 2011.

[59] X. Guo, J Fan, P. Schaumont, and I. Verbauwhede. Programmable and Parallel ECC Coprocessor Architecture: Tradeoffs between Area, Speed and Security. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2009)*, pages 289–303, Berlin/Heidelberg, 2009. Springer.

[60] X. Guo, S. Huang, L. Nazhandali, and P. Schaumont. Fair and Comprehensive Performance Evaluation of 14 Second Round SHA-3 ASIC Implementations. In *The Second SHA-3 Candidate Conference*, 2010.

[61] X. Guo, S. Huang, L. Nazhandali, and P. Schaumont. On The Impact of Target Technology in SHA-3 Hardware Benchmark Rankings. Cryptology ePrint Archive, Report 2010/536, 2010. `http://eprint.iacr.org/2010/536`.

[62] X. Guo, S. Huang, L. Nazhandali, and P. Schaumont. On The Impact of Target Technology in SHA-3 Hardware Benchmark Rankings. Cryptology ePrint Archive, Report 2010/536, 2010. `http://eprint.iacr.org/2010/536`.

[63] X. Guo and P. Schaumont. The Technology Dependence of Lightweight Hash Implementation Cost. In *ECRYPT Workshop on Lightweight Cryptography (LC2011)*, November 2011.

[64] X. Guo, M. Srivastav, S. Huang, D. Ganta, M. B. Henry, L. Nazhandali, and P. Schaumont. Performance Evaluation of Cryptographic Hardware and Software – Performance Evaluation of SHA-3 Candidates in ASIC and FPGA, May 2011. `http://rijndael.ece.vt.edu/sha3/`.

[65] X. Guo, M. Srivastav, S. Huang, D. Ganta, M. B. Henry, L. Nazhandali, and P. Schaumont. Silicon Implementation of SHA-3 Finalists: BLAKE, Grøstl, JH, Keccak and Skein. In *ECRYPT II Hash Workshop 2011*, May 2011.

[66] X. Guo, M. Srivistav, S. Huang, D. Ganta, M. B. Henry, L. Nazhandali, and P. Schaumont. Pre-silicon Characterization of NIST SHA-3 Final Round Candidates. In *14th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2011)*, 2011.

[67] X. Guo, M. Srivistav, S. Huang, D. Ganta, M. B. Henry, L. Nazhandali, and P. Schaumont. ASIC Implementations of Five SHA-3 Finalists. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 1 –6, March 2012.

[68] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In *Proceedings of Cryptographic Hardware and Embedded Systems*, pages 925–943, Berlin Heidelberg, January 2004. Springer.

[69] N. Gura, S. C. Shantz, H. Eberle, S. Gupta, V. Gupta, D. Finchelstein, E. Goupy, and D. Stebila. An End-to-End Systems Approach to Elliptic Curve Cryptography. In *CHES '02: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 349–365, London, UK, 2003. Springer-Verlag.

[70] J. Ha, J. Park, S. Moon, and S. Yen. Provably Secure Countermeasure Resistant to Several Types of Power Attack for ECC. In *Information Security Applications (WISA2007), LNCS4867*, pages 333–344, Berlin/Heidelberg, 2007. Springer.

[71] G. Hachez, F. Koeune, and J.-J. Quisquater. cAESar results: Implementation of Four AES Candidates on Two Smart Cards. In *In Second Advanced Encryption Standard Candidate Conference*, pages 95–108, 1999.

[72] D. Hankerson, A.J. Menezes, and S.A. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[73] L. Henzen, J.-P. Aumasson, W. Meier, and R. C.-W. Phan. VLSI Characterization of the Cryptographic Hash Function BLAKE. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(10):1746 –1754, 2011.

[74] L. Henzen, P. Gendotti, P. Guillet, E. Pargaetzi, M. Zoller, and F. Gürkaynak. Developing a Hardware Evaluation Method for SHA-3 Candidates. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *LNCS*, pages 248–263. 2010.

[75] C. Herbst and M. Medwed. Using Templates to Attack Masked Montgomery Ladder Implementations of Modular Exponentiation. In *Information Security Applications, WISA 2008*, pages 1–13, 2008.

[76] A. Hodjat, D. Hwang, L. Batina, and I. Verbauwhede. A hyperelliptic curve crypto coprocessor for an 8051 microcontroller. In *Proceedings of the 19th IEEE Workshop on Signal Processing Systems*, pages 93–98. IEEE, January 2005.

[77] N. Homma, A. Miyamoto, T. Aoki, A. Satoh, and A. Shamir. Collision-Based Power Analysis of Modular Exponentiation Using Chosen-Message Pairs. In *Cryptographic Hardware and Embedded Systems - CHES 2008*, pages 15–29, 2008.

[78] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.-S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 46–59. Springer Berlin / Heidelberg, 2006.

[79] D. Hwang, K. Tiri, A. Hodjat, B.-C. Lai, S. Yang, P. Schaumont, and I. Verbauwhede. AES-Based Security Coprocessor IC in 0.18- CMOS With Resistance to Differential Power Analysis Side-Channel Attacks. *Solid-State Circuits, IEEE Journal of*, 41(4):781 – 792, 2006.

[80] T. Itoh and S. Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) Using Normal Bases. In *Information and Computation*, pages 171–177, USA, 1988. Academic Press, Inc.

[81] T. Izu and T. Takagi. Exceptional Procedure Attack on Elliptic Curve Cryptosystems. In *Public Key Cryptography, PKC 2003*, pages 224–239, 2003.

[82] K. Järvinen and J. Skyttä. On Parallelization of High-Speed Processors for Elliptic Curve Cryptography. In *IEEE Trans. VLSI Systems*, pages 1162–1175, USA, 2008. IEEE.

[83] M. Joye. On the Security of a Unified Countermeasure. In *FDTC '08: Proceedings of the 2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 87–91, Washington, DC, USA, 2008. IEEE Computer Society.

[84] M. Joye and C. Tymen. Protections against Differential Analysis for Elliptic Curve Cryptography. In *CHES '01: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, pages 377–390, London, UK, 2001. Springer-Verlag.

[85] M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2002.

[86] E. Kavun and T. Yalcin. A Lightweight Implementation of Keccak Hash Function for Radio-Frequency Identification Applications. In Siddika Ors Yalcin, editor, *Radio Frequency Identification: Security and Privacy Issues*, volume 6370 of *Lecture Notes in Computer Science*, pages 258–269. Springer Berlin / Heidelberg, 2010.

[87] M. Kim, J. Ryou, and S. Jun. Efficient Hardware Architecture of SHA-256 Algorithm for Trusted Mobile Computing. In Moti Yung, Peng Liu, and Dongdai Lin, editors, *Information Security and Cryptology*, volume 5487 of *Lecture Notes in Computer Science*, pages 240–252. Springer Berlin / Heidelberg, 2009.

[88] M. Knezevic, K. Kobayashi, J. Ikegami, S. Matsuo, A. Satoh, U. Kocabas, J. Fan, T. Katashita, T. Sugawara, K. Sakiyama, I. Verbauwhede, K. Ohta, N. Homma, and T. Aoki. Fair and Consistent Hardware Evaluation of Fourteen Round Two SHA-3 Candidates. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, PP(99):1 –13, 2011.

[89] K. Kobayashi, J. Ikegami, M. Knezevic, X. Guo, S. Matsuo, Sinan Huang, L. Nazhandali, U. Kocabas, Junfeng Fan, A. Satoh, I. Verbauwhede, K. Sakiyama, and K. Ohta. Prototyping platform for performance evaluation of SHA-3 candidates. In *Hardware-Oriented Security and Trust (HOST), IEEE International Symposium on*, pages 60 –63, 2010.

[90] A. H. Koblitz, N. Koblitz, and A. Menezes. Elliptic Curve Cryptography: The Serpentine Course of a Paradigm Shift. Cryptology ePrint Archive, Report 2008/390, 2008.

[91] N. Koblitz. Elliptic curve cryptosystems. Mathematics of Computation. *Mathematics of Computation*, 48(177):203–209, January 1987.

[92] N. Koblitz. A Family of Jacobians Suitable for Discrete Log Cryptosystems. In *CRYPTO '88: Proceedings of the 8th Annual International Cryptology Conference on Advances in Cryptology*, pages 94–99, London, UK, 1990. Springer-Verlag.

[93] P.C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *CRYPTO'96:Advances in Cryptology*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[94] P.C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[95] O. Kömmerling and M.G. Kuhn. Design Principles for Tamper-Resistant Smartcard Processors. In *USENIX workshop on Smartcard Technology – SmartCard'99*, pages 9–20, 1999.

[96] M. Koschuch, J. Lechner, A. Weitzer, J. Großschädl, A. Szekely, S. Tillich, and J. Wolkerstorfer. Hardware/Software Co-design of elliptic curve cryptography on an 8051 microcontroller. In *Proceedings of Cryptographic Hardware and Embedded Systems*, pages 430–444, Berlin Heidelberg, January 2006. Springer.

[97] S. Kumar and C. Paar. Reconfigurable Instruction Set Extension for enabling ECC on an 8-bit Processor. In *Proceedings of Field Programmable Logic and Application*, pages 586–585, Berlin Heidelberg, January 2004. Springer.

[98] S. Kumar, T. Wollinger, and C. Paar. Optimum Digit Serial GF(2^m) Multipliers for Curve-Based Cryptography. *IEEE Transactions on Computers*, 55(10):1306–1311, 2006.

[99] I. Kuon and J. Rose. Measuring the Gap Between FPGAs and ASICs. 26(2):203 –215, 2007.

[100] X. Lai and J. L. Massey. A Proposal for a New Block Encryption Standard. In *Proceedings of EUROCRYPT '90 - Advances in cryptology, LNCS 473*, pages 389–404. Springer-Verlag, 1991.

[101] Y. W. Law, J. Doumen, and P. H. Hartel. Survey and Benchmark of Block Ciphers for Wireless Sensor Networks . *ACM Transactions on Sensor Networks (TOSN)*, 2(1):65 –93, 2006.

[102] G. Leander, C. Paar, A. Poschmann, and K. Schramm. New Lightweight DES Variants. In Alex Biryukov, editor, *Fast Software Encryption*, volume 4593 of *Lecture Notes in Computer Science*, pages 196–210. Springer Berlin / Heidelberg, 2007.

[103] P.-Y. Liardet and N. P. Smart. Preventing SPA/DPA in ECC Systems Using the Jacobi Form. In *CHES*, volume 2162 of *Lecture Notes in Computer Science*, pages 391–401. Springer, 2001.

[104] J. López and R. Dahab. Fast Multiplication on Elliptic Curves over GF(2^m) without Precomputation. In *CHES '99: Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, pages 316–327, London, UK, 1999. Springer-Verlag.

[105] 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? NIST 2nd SHA3 Candidates Conference, 2010.

[106] M. McLoone and J. McCanny. High Performance Single Chip FPGA Rijndael Algorithm Implementations. In *Cryptographic Hardware and Embedded Systems, CHES 2001*, volume 2162 of *LNCS*, pages 65 –76. Springer, 2001.

[107] M. Medwed and E. Oswald. Template Attacks on ECDSA. In *Information Security Applications, WISA 2008*, pages 14–27, 2008.

[108] Victor S. Miller. Use of Elliptic Curves in Cryptography. In *CRYPTO '85: Advances in Cryptology*, pages 417–426, London, UK, 1986. Springer-Verlag.

[109] P.L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.

[110] E. De Mulder, S. Örs, B. Preneel, and I. Verbauwhede. Differential power and electromagnetic attacks on a FPGA implementation of elliptic curve cryptosystems. *Computers & Electrical Engineering*, 33(5-6):367–382, 2007.

[111] A.H. Namin and M.A. Hasan. Hardware implementation of the compression function for selected SHA-3 candidates. CACR 2009-28, July 2009.

[112] NIST. CRYPTOGRAPHIC HASH ALGORITHM COMPETITION, May 2011. `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`.

[113] NIST. Third (Final) Round Candidates, December 2011. `http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html`.

[114] K. Okeya and K. Sakurai. Power Analysis Breaks Elliptic Curve Cryptosystems even Secure against the Timing Attack. In *INDOCRYPT*, volume 1977 of *Lecture Notes in Computer Science*, pages 178–190. Springer, 2000.

[115] M. O'Neill and M.J.B. Robshaw. Low-cost digital signature architecture suitable for radio frequency identification tags. *Computers Digital Techniques, IET*, 4(1):14 –26, january 2010.

[116] G. Orlando and C. Paar. A High Performance Reconfigurable Elliptic Curve Processor for GF(2^m). In *CHES '00: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, pages 41–56, London, UK, 2000. Springer-Verlag.

[117] F. Rodríguez-Henríquez, N. A. Saqib, A. Díaz-Pérez, and Ç. K. Koç. Optimum Digit Serial GF(2^m) Multipliers for Curve-Based Cryptography. *Computers, IEEE Transactions on*, 55(10):1306 –1311, 2006.

[118] R. Roman, C. Alcaraz, and J. Lopez. A Survey of Cryptographic Primitives and Implementations for Hardware-Constrained Sensor Network Nodes. 12(4):231 –244, 2007.

[119] Kumar. S., T. Wollinger, and C. Paar. Optimum Digit Serial GF(2^m) Multipliers for Curve-Based Cryptography. *Computers, IEEE Transactions on*, 55(10):1306 –1311, 2006.

[120] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede. Superscalar Coprocessor for High-Speed Curve-Based Cryptography. In *Proceedings of Cryptographic Hardware and Embedded Systems*, pages 415–429, Berlin Heidelberg, January 2006. Springer.

[121] A. Satoh and S. Morioka. Hardware-Focused Performance Comparison for the Standard Block Ciphers AES, Camellia, and Triple-DES. In *ISC 2003*, volume 2851 of *LNCS*, pages 252 –266. Springer, 2003.

[122] P. Schaumont, D. Ching, and I. Verbauwhede. An interactive codesign environment for domain-specific coprocessors. *ACM Trans. Des. Autom. Electron. Syst.*, 11:70–87, January 2006.

[123] P. Schaumont, D. Hwang, S. Yang, and I. Verbauwhede. Multilevel Design Validation in a Secure Embedded System. *IEEE Trans. Comput.*, 55:1380–1390, November 2006.

[124] P. Schaumont and I. Verbauwhede. A Component-Based Design Environment for ESL Design. *IEEE Design and Test of Computers*, 23(5):338–347, 2006.

[125] P. Schaumont and I. Verbauwhede. Hardware/Software Codesign for Stream Ciphers. SASC (State of the Art of Stream Ciphers), Special workshop hosted by the ECRYPT Network of Excellence in Cryptology, 2007.

[126] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata. The 128-Bit Blockcipher CLEFIA (Extended Abstract). In Alex Biryukov, editor, *Fast Software Encryption*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195. Springer Berlin / Heidelberg, 2007.

[127] D. Stebila and N. Thériault. Unified Point Addition Formulæ and Side-Channel Attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 354–368. Springer, 2006.

[128] Synopsys. Reference Methodology Retrieval System from Synopsys SolvNet, August 2010. `https://solvnet.synopsys.com/rmgen/`.

[129] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J.-M. Schmidt, and A. Szekely. High-Speed Hardware Implementations of BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, and Skein. Cryptology ePrint Archive, Report 2009/510, 2009. `http://eprint.iacr.org/2009/510`.

[130] S. Tillich, M. Feldhofer, M. Kirschbaum, T. Plos, J.-M. Schmidt, and A. Szekely. Uniform Evaluation of Hardware Implementations of the Round-Two SHA-3 Candidates. In *The Second SHA-3 Candidate Conference*, August 2010.

[131] K. Tiri and I. Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, pages 246–251. IEEE Computer Society, 2004.

[132] TOSHIBA. Toshiba CMOS Technology Roadmap for ASIC, May 2011. `http://www.toshiba-components.com/ASIC/Technology.html`.

[133] M. S. Turan, R. Perlner, L. E. Bassham, W. Burr, D. Chang, S. Chang, M. J. Dworkin, J. M. Kelsey, S. Paul, and R. Peralta. Status Report on the Second Round of the SHA-3 Cryptograhic Hash Algorithm Competition. NIST Interagency Report 7764, February 2011. `http://csrc.nist.gov/publications/nistir/ir7764/nistir-7764.pdf`.

[134] I. Verbauwhede and P. Schaumont. Design Methods for Security and Trust. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '07, pages 672–677, San Jose, CA, USA, 2007. EDA Consortium.

[135] C. D. Walter. Simple Power Analysis of Unified Code for ECC Double and Add. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 2004.

[136] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology CRYPTO 2005*, volume 3621 of *LNCS*. Springer, 2005.

[137] N. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective (3rd Edition)*. Addison-Wesley, 2004.

[138] D. Wheeler and R. Needham. TEA, a Tiny Encryption Algorithm. In *FSE94, LNCS 1008*, pages 97–110. Springer-Verlag, 1994.

[139] J. Worley, B. Worley, T. Christian, and C. Andworley. AES Finalists on PA-RISC and IA-64: Implementations and Performance. In *In Third Advanced Encryption Standard Candidate Conference*, 2001.

[140] L. Xiao and M. H. Heys. Hardware Performance Characterization of Block Cipher Structures. In *CT-RSA 2003*, volume 2612 of *LNCS*, pages 176 –192. Springer, 2003.

[141] S.-M. Yen and M. Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Trans. Computers*, 49(9):967–970, 2000.

[142] S.-M. Yen, S. Kim, S. Lim, and S. Moon. A Countermeasure against One Physical Cryptanalysis May Benefit Another Attack. In *Proceedings of the 4th International Conference Seoul on Information Security and Cryptology*, ICISC '01, pages 414–427, London, UK, 2002. Springer-Verlag.

[143] S.-M. Yen, L.-C. Ko, S.-J. Moon, and J. Ha. Relative Doubling Attack Against Montgomery Ladder. In *Information Security and Cryptology, ICISC 2005*, 2005.