Energy Efficient Hardware Design for Securing the Internet-of-Things

by

Yiqun Zhang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in the University of Michigan
2018

Doctoral Committee:

Professor Dennis M. Sylvester, Chair
Professor David T. Blaauw
Assistant Professor Hun Seok Kim
Associate Professor Kenn R. Oldham

Yiqun Zhang

zhyiqun@umich.edu

ORCID iD: 0000-0001-5584-6503

ACKNOWLEDGMENTS

These past seven years at the University of Michigan, from college to Ph.D, have shaped my life, and I would like to express my gratitude to the people with whom I shared this amazing journey.

First, I would like to thank my Ph.D advisor Dennis Sylvester. I took all my circuit classes with him during my undergraduate study and became interested in this field. As his Ph.D student, he guided me to think critically and gave me a lot of freedom to pursue my research interests and develop my career. Second, I would like to thank Professor David Blaauw, who always gave great advice during meetings and taught me to always challenge myself. Also, thanks Professors Hunseok Kim and Kenn Oldham for serving on my dissertation committee and providing guidance and advice on my thesis. In addition, I would like to thank Professor Zhengya Zhang, who mentored me during the summer of my junior year on a circuits research project. He was always patient and very helpful in answering my questions.

I am grateful to have had some incredible mentors from industry and other institutions. First, it was a pleasure to meet Edith Beigne at conferences, and I would like to thank her for organizing women-in-circuits events and giving me a lot of technical advice. Also, thanks to Doug Gardner, Junhua Shen and Sam Fuller of Analog Devices for their collaboration and support. Finally, thanks to Brucek Khailany and Yanqing Zhang from NVidia, and Xiaolin Lu and Thomas Tsai from Texas Instruments for their mentorship during my internships.

I feel honored to be in a very competitive research group. In particular, I would like to thank Mahmood Khayatzadeh who guided me through my first year of Ph.D and equipped me with lots of research skills. Special thanks to Kaiyuan Yang, Supreet Jeloka, Qing Dong

and Yejoong Kim for the stimulating discussions and insightful suggestions. Thanks all my fellow labmates for their help and the sleepless nights we spent together which made tape-outs and paper submission deadlines much more fun: Li, Inhee, Laura, Dave, Jingcheng, Ziyun, Jongyup, Jeongsup, Minchang, Sechang, Kyojin, Li-Xuan, Zhen, Taekwang, Wanyeong, Taewook, Wootaek, Jihwan, Xiao, Zhehong, Yao, Myungjoon, Zhiyoong, Seok-Hyeon and Gyouho. In addition, I would like to thank Fran Doman and Sarah Towler for managing and organizing lab events, particularly for preparing tasty food at lunch meetings.

I would like to thank my parents for teaching me to be independent and for their endless support no matter how far away I am from home. Also, a special thanks to my boyfriend Sam Xi for his love and care. I learned from him to pet every dog I meet and enjoy every moment of life. Last but definitely not least, thanks to my friends Chunyang, Shuanghong, Xiaozhou, Wenzhe, Lin, Chen, Xiaoyu and Xuefei, for spending these years with me in the beautiful small town of Ann Arbor.

TABLE OF CONTENTS

ACKNOWLEDGMENTS ii

LIST OF FIGURES vii

LIST OF TABLES ix

ABSTRACT x

CHAPTER

I. Introduction 1

1.1 Internet of Things (IoT) 1
1.2 IoT Security 3
1.3 Design Principles 3
1.4 Dissertation Outline 4

II. Theory and Background 6

2.1 Security Primitives 6
2.2 Elliptic Curve Cryptography 8
2.2.1 Field Arithmetic 8
2.2.2 Elliptic Curve Arithmetic 10
2.2.3 Elliptic Curve Protocols 13
2.3 Advanced Encryption Standard (AES) 13
2.4 Authenticated Encryption with associated data (AEAD) 15
2.4.1 Galois/Counter Mode (GCM) 15
2.5 Keccak Hash Function 17
2.6 Existing Secure ICs 17

III. A Compact AES Accelerator for Mobile SoCs and IoT 20

3.1 Previous Work 20
3.2 Baseline implementation 21

3.3 Proposed Energy Efficient AES . . . 22
3.3.1 DataReg & ShiftRow . . . 22
3.3.2 KeyReg . . . 25
3.3.3 Sbox . . . 26
3.4 Testchip and Measurements . . . 28
3.5 Conclusion . . . 32

IV. Recryptor: A Reconfigurable Cryptographic Cortex-M0 Processor with In-Memory and Near-Memory Computing . . . 33

4.1 Motivation and Previous Works . . . 34
4.2 Proposed Recryptor Overview . . . 35
4.3 Recryptor's In-Memory Computing . . . 37
4.3.1 10T Bitcell and Accelerated Bitwise Operations . . . 37
4.3.2 The Configuration of Bank Division . . . 38
4.4 Recryptor's Near-Memory Computing . . . 38
4.4.1 Shifter . . . 38
4.4.2 Rotator . . . 39
4.4.3 Sbox . . . 40
4.5 Programmability and Optimized Algorithm Implementations . . . 42
4.5.1 Finite Field Multiplication and Reduction . . . 42
4.5.2 AES . . . 44
4.5.3 Keccak-f . . . 45
4.5.4 Cryptographic Finite State Machines . . . 46
4.6 Testchip and Measurements . . . 46
4.6.1 Testchips and Measurements . . . 46
4.6.2 Results Among Algortihms and Comparisons . . . 50
4.7 Discussion . . . 52
4.7.1 Recryptor Optimization . . . 52
4.7.2 Recryptor vs. Near-memory-computing-Only . . . 53
4.7.3 Power Analysis Attack . . . 55
4.8 Conclusion . . . 57

V. Recryptor M-ulator: A Simulator for In-Memory Computation . . . 58

5.1 Motivation and Background . . . 58
5.2 Proposed Framework . . . 59
5.3 Case study: Elliptic Curve Arithmetic . . . 60
5.3.1 Field Arithmetic . . . 60
5.3.2 Curve Arithmetic . . . 61
5.4 Case study: AES-GCM . . . 61
5.5 Conclusion . . . 62

VI. iRazor: A Current-Based Error Detection and Correction Scheme for PVT variation . . . 64

6.1 Motivation 65
6.2 Previous Works 65
6.3 Proposed iRazor Circuit and Analysis 67
6.3.1 The iRazor Flip-flop 67
6.3.2 Analysis of Robustness, Area and Energy 71
6.4 Error Detection and Correction Scheme 73
6.5 Automated iRazor Design Flow 74
6.6 Testchip 79
6.7 Comparisons with Binning and Canary Techniques 80
6.8 Experimental Results and Overall Comparison 85
6.9 Conclusion 87

VII. Conclusions and Future Directions 90

7.1 Contributions 90
7.2 Future Directions 91

BIBLIOGRAPHY 92

LIST OF FIGURES

Figure

1.1 Bell's law of computer scaling and computer class [68, 74] 2
1.2 Internet of Things 2
1.3 Energy/bit of IoT building blocks (Adapted from [76]) 4
2.1 Elliptic curve point addition 11
2.2 AES Sbox example 14
2.3 AES ShiftRows Operation 15
2.4 Authenticator WorkScheme 19
3.1 AES 21
3.2 Optimized ShiftRow [46] 23
3.3 Proposed DataReg & ShiftRow Technique 23
3.4 Detailed hardwired solution 24
3.5 One-hot shift encoding 25
3.6 Proposed Sbox datapath in GF($(2^4)^2$) with 2 cycles 27
3.7 AES Chip micrographs 28
3.8 AES baseline implementation 29
3.9 Measured performance across voltages of the proposed AES design 29
3.10 Measured energy across voltages of the proposed AES design 30
3.11 Power breakdown of baseline and proposed AES designs 30
4.1 Proposed Recryptor architecture. 35
4.2 Proposed Crypto-SRAM Bank (CSB). 36
4.3 10T bitcell and supported bitwise operations on two words. 37
4.4 The subbank configuration and implementation. 39
4.5 Shifter design 40
4.6 Arbitrary 64-bit Rotator. 41
4.7 Proposed SBOX implementation. 41
4.8 Die photo of baseline and Recryptor in 40nm CMOS. 47
4.9 Measured Fmax of the custom 10T SRAM. 48
4.10 Measured Vmin across temperature of the custom 10T SRAM. 48
4.11 Measured frequency of Baseline and Recryptor across voltages. 49
4.12 Measured power of Baseline and Recryptor across voltages. 49
4.13 Simulated power breakdown of different security functions. 50

4.14 Comparison of an ASIC [81], Recryptor, Baseline and a coprocessor [64] design for AES. . . . 52
4.15 Simulated DPA on an AES ASIC with 20 traces. . . . 56
4.16 Simulated DPA on Recryptor's CSB with 300 traces. . . . 57
6.1 Schematic of the proposed iRazor flip-flop with error detection capability, and its energy, delay and area compared with conventional flip-flop standard cell (both positive edge triggered) . . . 68
6.2 Waveforms in iRazor flip-flop . . . 69
6.3 Statistical analysis of a) virtual ground voltage under errors vs. T_{FR} (1,000 Monte Carlo runs). Whiskers indicate 3 standard deviations around the mean value . . . 72
6.4 Overall iRazor EDAC scheme diagram . . . 73
6.5 (a)iRazor timing diagram; (b) timing analysis of the error critical path . . 75
6.6 Architecture-independent automated flow for iRazor flip-flop replacement and clustering . . . 76
6.7 Design complexity (in number of iRazor flip-flops) vs. targeted timing slack of iRazor . . . 77
6.8 Path delay histogram of baseline and iRazor design . . . 77
6.9 iRazor cluster spatial position within the on-die processor footprint . . . 78
6.10 iRazor effective overhead explicit calculation . . . 78
6.11 Die photo of baseline and iRazor Cortex-R4 processor in 40nm CMOS . . . 79
6.12 (a) Detailed frequency histogram and margin analysis of baseline at 1V; (b) Baseline frequency margin across 0.6 1V voltage range, including 10% voltage margin, 60°C temperature margin, 3 sigma process margin. The frequency margin is normalized to the average across dice of its actual frequency at nominal voltage/temperature conditions. . . . 80
6.13 Detailed frequency histogram and margin analysis of frequency binning method at 1V . . . 81
6.14 Margin histogram for different methods (1V, room temperature). . . . 82
6.15 Fitting of operating frequency vs. ring oscillator frequency in simple canary fitting method . . . 83
6.16 Detailed frequency histogram and margin analysis of simple canary method at 1V. . . . 84
6.17 Fitting of processor frequency vs. ring oscillator frequency for T/V-specific canary at 25°C and 85°C. . . . 84
6.18 iRazor frequency at point of first failure (PoFF) vs. optimal frequency across voltages. . . . 85
6.19 Performance comparison between the margined iRazor and other methods across 0.6-1V voltage range. . . . 86
6.20 Power comparison with the margined iRazor across 0.6 1V voltage range. . 87

LIST OF TABLES

Table

2.1	Key length of ECC and RSA for different security levels	7
2.2	The number of operations for different coordinate systems (I: field inversion, M: field multiplication)	11
2.3	Authenticator Summary	19
3.1	Comparison table of DataReg & ShiftRow	25
3.2	Comparison table of different Sbox implementations	27
3.3	Chip measurement summary and comparison table of AES designs	31
4.1	Shifter Supported Functions	40
4.2	Estimated # operations according to Algorithm 1.	43
4.3	Enabled sub-bank of different word length for FFM.	44
4.4	Comparison table of different crypto algorithms and designs.	51
4.5	The effective memory capacity for computing and the required modules.	53
4.6	# Cycles to run Binary Elliptic Curve Finite Field Multiplication and Reduction	54
4.7	Calculated Area Overhead of Recryptor and Near-memory-computing (NMC)	55
5.1	Number of cycles for Finite Field Multiplication and Reduction	60
5.2	Number of cycles on $\mathbb{F}_{2^{163}}$ Koblitz curve	61
5.3	Number of cycles on $\mathbb{F}_{2^{233}}$ Koblitz curve	61
5.4	Number of cycles on $\mathbb{F}_{2^{283}}$ Koblitz curve	62
5.5	Number of cycles on $\mathbb{F}_{2^{409}}$ Koblitz curve	62
5.6	Number of cycles for AES- GCM	63
6.1	Comparison table of EDAC approaches and iRazor.	88

ABSTRACT

The Internet of Things (IoT) is a rapidly growing field that holds potential to transform our everyday lives by placing tiny devices and sensors everywhere. The ubiquity and scale of IoT devices require them to be extremely energy efficient. Given the physical exposure to malicious agents, security is a critical challenge within the constrained resources. This dissertation presents energy-efficient hardware designs for IoT security.

First, this dissertation presents a lightweight Advanced Encryption Standard (AES) accelerator design. By analyzing the algorithm, a novel method to manipulate two internal steps to eliminate storage registers and replace flip-flops with latches to save area is discovered. The proposed AES accelerator achieves state-of-art area and energy efficiency.

Second, the inflexibility and high Non-Recurring Engineering (NRE) costs of Application-Specific-Integrated-Circuits (ASICs) motivate a more flexible solution. This dissertation presents a reconfigurable cryptographic processor, called *Recryptor*, which achieves performance and energy improvements for a wide range of security algorithms across public key / secret key cryptography and hash functions. The proposed design employs circuit techniques in-memory and near-memory computing and is more resilient to power analysis attack. In addition, a simulator for in-memory computation is proposed. It is of high cost to design and evaluate new-architecture like in-memory computing in Register-transfer level (RTL). A C-based simulator is designed to enable fast design space exploration and large workload simulations. Elliptic curve arithmetic and Galois counter mode are evaluated in this work.

Lastly, an error resilient register circuit, called *iRazor*, is designed to tolerate unpredictable variations in manufacturing process operating temperature and voltage of VLSI

systems. When integrated into an ARM processor, this adaptive approach outperforms competing industrial techniques such as frequency binning and canary circuits in performance and energy.

CHAPTER I

Introduction

1.1 Internet of Things (IoT)

Following the rapid trend of semiconductor industry over the past 50 years, Moore's Law has accurately predicted the single-chip evolution of doubling the number of transistors every two years. While Moore's Law slows down, Bell's Law [10] emphasis more on the computer classes, saying that a new computer class forms approximately every decade, as shown in Figure 1.1. As electronic devices are getting smaller and smaller, these embedded processors or small sensors can be integrated into different systems for numerous applications to transform the everyday lives of people, such as autonomous cars, smart city and smart home. Such kind of system is usually called the Internet of Things (IoT).

The *Internet* is used to send and receive information for the communication of different objects, or *Things* (Figure 1.2). Although a general purpose computer can achieve most of these functions, we don't want to carry a laptop anywhere. The idea of the Internet of Things suggests that rather than having few powerful computing devices, instead, it's preferable to have a larger number of devices which could be less powerful to assist with our lives.

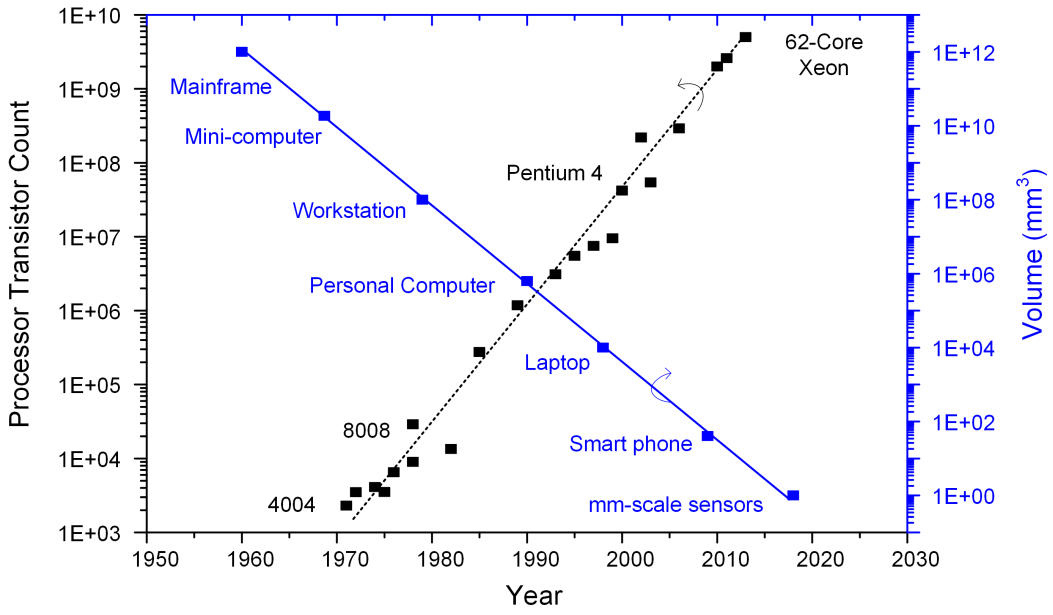


Figure 1.1: Bell's law of computer scaling and computer class [68, 74]

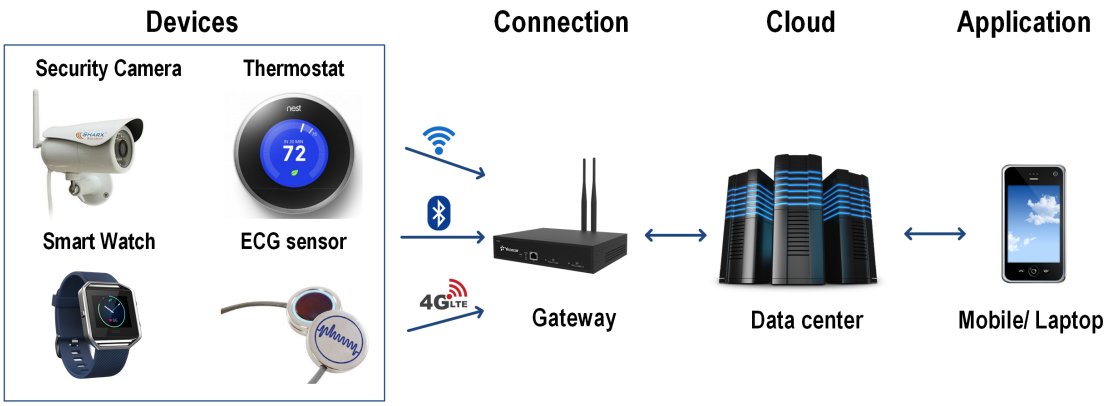


Figure 1.2: Internet of Things

1.2 IoT Security

Security is of utmost concern for IoT applications due to the potential pervasiveness of IoT devices. Classical system security for distributed systems and databases also applies to the IoT systems. For example, an adversary can eavesdrop on the communication, analyze the network behavior, then inject false messages or replay old messages to attack the network. He can also do the denial-of-service (DoS) attack to jam the wireless channel and violate the availability of networks. In addition to these classical attacks, sensor networks are especially vulnerable to *physical attacks*, due to the direct access to the sensor node hardware for attackers.

As some real world attack examples, a Jeep Cherokee is hacked by researchers in 2015, through the messages on the local network of the car over a cellular connection [30, 48]. Another example is that researchers use a smart lamp connected to the network to create a ZigBee Chain Reaction, and they make the statement that these attacks can get catastrophic results by naming it "IoT Goes Nuclear" [24].

In order to ensure secure communications, IoT networks require authentication for trustworthiness, data encryption for confidentiality and integrity, and fault tolerance for resilient operation under attack.

1.3 Design Principles

The security frameworks for sensor networks need to satisfy the following requirements

- **Energy Efficiency:** Energy efficiency is one of the most important criteria for designing IoT devices, since most of them operate on the limited battery life. For example, an IoT network example - I3Mote [45] reports to sustain 1.4 years of operation with 2 AAA batteries, while it needs to explore low power techniques in order to target 10 years of operation for industrial IoT applications.

- **Lightweightness:** Since sensor nodes usually have constraint resources, a major chal-

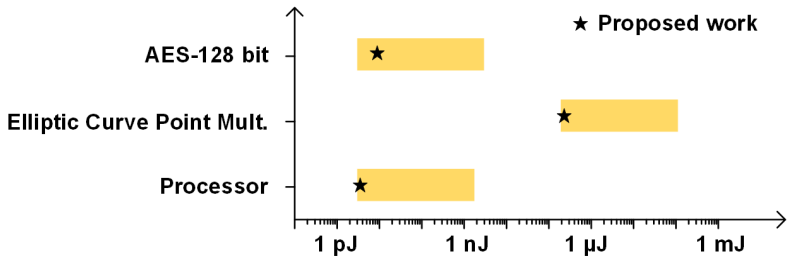


Figure 1.3: Energy/bit of IoT building blocks (Adapted from [76])

lenge for IoT security is to implement the security primitives efficiently (i.e. less power/energy, computational time and area), while still maintains security strength.

- **Flexibility:** Security standards will evolve over time. Also, depending on different applications, the IoT system requires different standards and level-of security.

- **Scalability:** The network should have minimum overhead and is functionally not affected by adding and deleting nodes.

- **Performance:** There is a low latency requirement for control-oriented and mission-critical applications, for example in a driverless car which captures sensor data to control steering wheels.

1.4 Dissertation Outline

The goal of this thesis is to design energy efficient hardware for IoT security. Figure 1.3 shows the energy per bit cost of three IoT building blocks to be optimized and their results within the range of state-of-art works. The proposed designs include an accelerator for symmetric key cipher, a crypto-coprocessor for asymmetric key cryptography, and in-situ variation detection technique for general purpose processor.

The remainder of this disseration is structured into the following chapters, which will explore the related topics in detail.

Chapter 2 discusses the theory of security algorithms covered in this thesis, including advanced encryption standard (AES), elliptic curve cryptography (ECC), authenticated en-

cryption with associated data (AEAD) and Keccak function. In addition, some existing secure IC examples are shown in this section.

Chapter 3 proposes an energy efficient and low cost implementation of the AES algorithm in 40nm CMOS [81]. The proposed design eliminates the ShiftRow stage in conventional AES implementations, replaces flip-flops in data and key storage with latches using re-timing, uses a 2-stage Sbox in native $GF(2^4)^2$ composite-field computation and glitch reduction techniques.

Chapter 4 proposes a reconfigurable cryptographic processor, called Recryptor [79], which exploits in-memory (with custom 10T bitcell) and near-memory computing (custom shifter, rotator and sbox designs) to achieve energy efficiency, performance, and programmability for IoT security applications. We demonstrate Recryptor's programmability by implementing of the cryptographic primitives of various public/secret key cryptography and hash functions.

Chapter 5 provides a simulator design for in-memory computing, to shorten the evaluation time for large workloads. We demonstrate the speedup for elliptic curve arithmetic which can take up to multi-million cycles and AES based galois counter mode.

Chapter 6 proposes a lightweight error detection and correction approach, called iRazor [77], to suppress the cycle time margin that is traditionally added to VLSI systems to tolerate process, voltage and temperature variations. iRazor is based on a novel current-based detector requiring only 3 additional transistors, which is embedded in flip-flops on potentially critical paths. The proposed scheme is implemented in an ARM Cortex-R4 microprocessor in 40nm through an automated iRazor flip-flop insertion flow. iRazor is also compared to other popular techniques that mitigate the impact of variations.

CHAPTER II

Theory and Background

This chapter discusses the security algorithms covered in this thesis, ranging from Elliptic Curve Cryptography, Advanced Encryption Standard, to Galois Counter Mode and Keccak function. In addition, we introduce some existing secure ICs from industrial products implementing these algorithms.

2.1 Security Primitives

Cryptographic primitives provide the insurance for basic functional security and are foundations to create secure protocols. Those security primitives can be divided into three types:

- **Secret Key Cryptography (SKC):** shares a single secret key between the two communicating peers for both encryption and decryption. SKC provides data *confidentiality*. For example, the Advanced Encryption Standard (AES) is a commonly used symmetric block cipher, approved by NIST in 2001 [53]. By using mode of operation, SKC can also provide integrity and authentication. These algorithms are usually easy to implement.

- **Public Key Cryptography (PKC):** uses a key pair - private keys to be kept private and public keys which is publicly known. Any operation done with a private key can only be reversed with the paired public key, and vice versa. PKC is useful for

authentication, while the computation cost is high. The well-known RSA crypto-system was proposed by Ron Rivest, Adi Shamir and Len Adleman in 1977 [63]. Elliptic Curve Cryptography (ECC) [41, 49] was discovered in 1985 and is becoming a popular option for PKC; for example, the STSAFE-A100 chip uses ECC for authentication [69]. For the desired security level, ECC needs significantly smaller keys than RSA, as shown in Table 2.1 [63], which results in smaller energy and memory requirements.

Table 2.1: Key length of ECC and RSA for different security levels

Security Level (bits)	**Elliptic Curve (bits)**	**RSA (bits)**
80 (SKIPJACK)	160	1024
112 (Triple-DES)	224	2048
128 (AES-Small)	256	3072
192 (AES-Medium)	384	8192
256 (AES-Large)	512	15360

- **Hash primitives:** provide a 'digital fingerprint' of the data as a hash value, which is used to detect changes. Hash functions irreversibly 'encrypt' information by compressing a set of data of variable length into a set of fixed length data. Hash functions assure the *integrity* of the information flow, providing a unique fingerprint in the form of a Message Authentication Code (MAC). MAC is usually computed using SKC with a special mode of operation called CBC-MAC. The Keccak hash function won the SHA-3 competition hosted by NIST in 2012 [27].

Typically, public (or asymmetric) key cryptography is used for key exchange, and symmetric (or secret) key cryptography is used for efficiently encrypting data. Hash function is used for message integrity.

2.2 Elliptic Curve Cryptography

Elliptic curves have been studied by mathematicians for more than a hundred years to solve a wide range of problems. In 1985, Neal Koblitz and Victor Miller independently designed public-key cryptography based upon elliptic curves. Three kinds of fields - prime fields, binary fields, and optimal extension fields - are good for efficient implementations of elliptic curve systems. In this research, we mainly target at binary Koblitz curves.

Finite fields of order 2^m are called binary fields and the elements are the binary polynomials of degree at most $m-1$.

$$\mathbb{F}_{2^{\mathrm{m}}} = a_{\mathrm{m\text{-}1}} z^{\mathrm{m\text{-}1}} + a_{\mathrm{m\text{-}2}} z^{\mathrm{m\text{-}2}} + \cdots + a_2 z^2 + a_1 z^1 + a_0, a_{\mathrm{i}} \in \{0, 1\}$$

2.2.1 Field Arithmetic

Field Addition

The addition of two binary field polynomials is performed as bitwise exclusive or (XOR).

Field Multiplication

Algorithm 1 López-Dahab left-to right comb method with windows of width ω

Input $x = (x_{\mathrm{m\text{-}1}}, \ldots, x_0)_2$, $y = (y_{\mathrm{m\text{-}1}}, \ldots, y_0)_2$
Output $c = xy = (c_{2\mathrm{m\text{-}1}}, \ldots, c_0)_2$

1: Compute $T(\mu) \leftarrow \mu y$ for all polynomials μ of degree at most $\omega - 1$
2: $C \leftarrow 0$
3: **for** $j \leftarrow \lceil m/\omega - 1 \rceil$ down to 0 **do**
4: $\quad \mu = (\ \mu_{\omega-1}\ , \ldots,\ \mu_0\ ) = (x >> j \cdot \omega)$'s lowest ω bits
5: $\quad$ Add T_μ to C
6: $\quad$ **if** $j \neq 0$ **then**
7: $\quad\quad C \leftarrow C \cdot z^\omega$
8: $\quad$ **end if**
9: **end for**
10: **return** c

An optimized method of polynomial multiplication is the López-Dahab (LD) left-to right comb method with windows of width ω (see Algorithm 1).

Field Squaring

If $a(z) = a_{\text{m-1}} z^{\text{m-1}} + \cdots + a_2 z^2 + a_1 z^1 + a_0$, then

$$a(z)^2 = a_{\text{m-1}} z^{\text{2m-2}} + \cdots + a_2 z^4 + a_1 z^2 + a_0$$

Field Reduction

For either field multiplication or squaring, the results have degree at most $2m-2$. Then, the result $c(z)$ needs to be reduced back to degree of at most $m-1$. For any arbitrary reduction polynomials $f(z) = z^m + r(z)$, where $r(z)$ is a binary polynomial of degree at most $m-1$, reduction $c(z) modulo f(z)$ can be done one bit at a time from the leftmost bit.

In addition, fast reduction has been proposed with $f(z)$ being a trinomial or a pentanomial with middle terms close to each other. Data is processed with the unit of words (commonly 32-bit or 64-bit) in modern processors, therefore, reduction can be efficiently performed one word at a time. For the fast reduction, the following reduction polynomials are recommended by NIST in the FIPS 186-4 standard [55]:

$$f(z) = z^{163} + z^7 + z^6 + z^3 + 1$$
$$f(z) = z^{233} + z^{74} + 1$$
$$f(z) = z^{283} + z^{12} + z^7 + z^5 + 1$$
$$f(z) = z^{409} + z^{87} + 1$$
$$f(z) = z^{571} + z^{10} + z^5 + z^2 + 1$$

An example of fast reduction modulo of 233-bit with word (W=32) is shown in Algorithm 2.

Field Inversion

The inverse of a nonzero element $a \in \mathbb{F}_{2^m}$ is an unique element $g \in \mathbb{F}_{2^m}$, such that $ag = 1$ in $\mathbb{F}_{2^m}$. The inverse can be efficiently computed by the extended Euclidean algorithm.

Algorithm 2 Fast reduction modulo $f(z) = z^{233} + z^{74} + 1$ (W=32)

Input: A binary polynomial $c(z)$ of degree at most 464
Output: $c(z)$ *mod* $f(z)$

1: **for** i from 15 downto 8 **do**
2: $T \leftarrow C[i]$
3: $c[i-8] \leftarrow C[i-8] \bigoplus (T << 23)$
4: $c[i-7] \leftarrow C[i-7] \bigoplus (T << 9)$
5: $c[i-5] \leftarrow C[i-5] \bigoplus (T << 1)$
6: $c[i-4] \leftarrow C[i-4] \bigoplus (T >> 31)$
7: **end for**
8: $T \leftarrow C[7] >> 9$
9: $C[0] \leftarrow C[0] \bigoplus T$
10: $C[2] \leftarrow C[2] \bigoplus (T << 10)$
11: $C[3] \leftarrow C[3] \bigoplus (T >> 22)$
12: $C[7] \leftarrow C[7] \& 0x1FF$
13: **Return** $(C[7], C[6], C[5], C[4], C[3], C[2], C[1], C[0])$

2.2.2 Elliptic Curve Arithmetic

Point addition and doubling

Assume that $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are two points on the elliptic curve, and $R = P + Q$. To get R, we need to draw a line through point P and Q, then extend it in both directions. The intersection of this line between the curve is at point R', and the final result R can be found by reflecting R' around the x-axis, as shown in Figure 2.1.

For point doubling, i.e. $P = Q$, we draw a line to be the tangent of the curve at point P. Similarly as point addition, the result is found by reflecting the intersection point of this line and the curve.

Coordinate systems

In the standard affine coordinate system, both the point addition and doubling require some field inversions, which are very expensive. It's common to represent the point in another coordinate system, where no inversions are needed during calculation and there only needs two inversions to transfer to the new system and back.

• **Affine coordinates (A)** The affine coordinate (x, y) represents a point on the elliptic

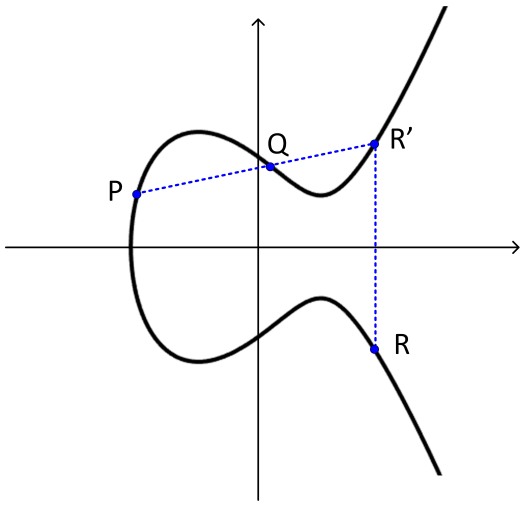


Figure 2.1: Elliptic curve point addition

Table 2.2: The number of operations for different coordinate systems (I: field inversion, M: field multiplication)

Coordinate system	Doubling	General Addition	Mixed Addition
Affine	1I + 1M	1I + 1M	-
Projective	7M	13M	12M (P+A)
Jacobian	5M	14M	10M (J+A)
López-Dahab	4M	14M	8M (LD+A)

curve.

- **Standard Projective coordinates (P)** The standard projective point (X, Y, Z), $Z \neq 0$ corresponds to the affine point $(X/Z, Y/Z)$.

- **Jacobian Projective coordinates (J)** The Jacobian projective point (X, Y, Z), $Z \neq 0$ corresponds to the affine point $(X/Z^2, Y/Z^3)$.

- **López-Dahab coordinates (LD)** The López-Dahab point (X, Y, Z), $Z \neq 0$ corresponds to the affine point $(X/Z, Y/Z^2)$.

The number of operations required to perform a point addition and point doubling for these four coordinates are shown in Table 2.2.

Point multiplication

Point multiplication is the operation of multiplying a scalar k with a point P on the elliptic curve.

$$kP = P + P + \cdots + P \ ; ((k-1) \text{ additions})$$

Random point addition means that P is random; while fixed point addition is that P is fixed. The most basic algorithm for random point addition is the double-and-add method (see Algorithm 3). And the expected number of running time of this algorithm is approximately $m/2$ point additions (A) and m point doublings (D), as:

$$\tfrac{m}{2}A + mD$$

Algorithm 3 Double-and-add method for point multiplication in $\mathbb{F}_{2^m}$

Input: $k = (k_{m-1}, \ldots, k_0)_2$, $P \in E(\mathbb{F}_{2^m})$
Output: kP

1: $Q \leftarrow \infty$
2: **for** i from 0 to $m-1$ **do**
3: **if** $k_i = 1$ **then**
4: $Q \leftarrow Q + P$
5: **end if**
6: $P \leftarrow 2P$
7: **end for**
8: **Return** Q

Koblitz curves

Koblitz curves are defined over $\mathbb{F}_2$ as follows. The main advantage of these curves is that point multiplication algorithms don't need to use any point doublings.

$$E_0 : y^2 + xy = x^3 + 1$$
$$E_1 : y^2 + xy = x^3 + x^2 + 1$$

τ-adic non-adjacent form (TNAF)

For any positive integer k, it can be written in the form $k = \sum_{i=0}^{l-1} \mu_i \tau^i$ where each $\mu_i \in 0, \pm 1$. This τ-adic non-adjacent representation can be calculated by repeatedly dividing k by τ.

With the τ-adic representation for k, it would have a small number of nonzero digits, which results in decreasing the number of point additions. This TNAF method for point multiplication on Koblitz curves (see Algorithm 4) has an expected running time of

$$\frac{m}{3}A$$

Algorithm 4 TNAF method for point multiplication on Koblitz curves

Input: Integer $k \in [1, n-1]$, $P \in E(\mathbb{F}_{2^m})$ of order n
Output: kP

1: Compute $\rho' = k$ partmod δ
2: Compute TNAF$(\rho') = \sum_{i=0}^{l-1} \mu_i \tau^i$
3: $Q \leftarrow \infty$
4: **for** i from $l-1$ downto 0 **do**
5: $\quad Q \leftarrow \tau Q$
6: $\quad$ **if** $\mu_i = 1$ **then** $Q \leftarrow Q + P$
7: $\quad$ **end if**
8: $\quad$ **if** $\mu_i = -1$ **then** $Q \leftarrow Q - P$
9: $\quad$ **end if**
10: **end for**
11: **Return** Q

2.2.3 Elliptic Curve Protocols

The Ellpitic Curve Discrete Logarithm Problem (ECDLP) is the fundamental for Elliptic Curve Cryptosystems, including Elliptic Curve Diffe Hellman (ECDH) and Elliptic Curve Digital Signature Algorithm (ECDSA). ECDH is a key agreement protocol. ECDSA is for signature generation and verification. More details can be found at [33].

2.3 Advanced Encryption Standard (AES)

In 2001, the national institute of standards and technology selected AES, which is short for advanced encryption standard, as the new symmetric key cipher standard. AES is a

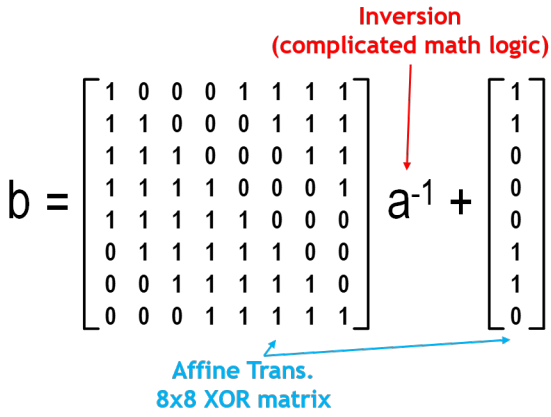


Figure 2.2: AES Sbox example

widely-used block cipher algorithm for symmetric encryption in a large range of applications.

The number of rounds of AES is 10 for the 128-bit encryption key, 12 for 192-bit, and 14 for 256-bit. For encryption, each round consists of the following four steps:

- **SubBytes** is a byte substitution. Sbox is a nonlinear byte substitution function (Figure 2.2). The input is an 8-bit data in galois field GF(2^8). The operation of multiplicative inversion is applied first. Then an affine transformation is applied, which means multiply an 8-by-8 matrix and add an 8-by-1 array. In this galois field, addition is bit-wise XOR and multiplication is bit-wise AND.

- **ShiftRows** means the bytes are being shifted within each row. For AES-128bits, the 16 bytes of data is shown in the 4-by-4 matrix format Figure 2.3. In more detail, the 4 bytes in the 1st row of this matrix will shift left by 0 position, which means they stay in the same location in the ShiftReg. The bytes in the 2nd row will shift left by 1. Similar operations apply to the 3rd and 4th rows, which will shift left by 2 and 3 positions.

- **MixColumns** is an operation that a predetermined matrix will multiply the data in the ShiftReg and the result is stored in the MixColumn register. Based upon matrix multiplication, the original data bytes D15, D10, D5 and D0 in the first column will perform multiplication and addition, with the first row of this predetermined matrix

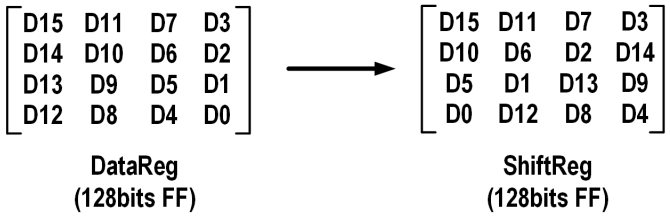


Figure 2.3: AES ShiftRows Operation

and to produce the new data byte D15. Similar operation is performed to get the new data byte D14, D13 and D12. So the first column in ShiftReg is the input to the calculation to produce the first column data in the MixColReg. Therefore, MixColumn operation retrieves 4 bytes of data as inputs and outputs 4 bytes of data every 4 cycles.

- **AddRoundKey** is the last step that data would be XORed with the round key that is generated at each iteration.

2.4 Authenticated Encryption with associated data (AEAD)

There have been practical attacks due to the separation of confidentiality and authentication of block cipher operation modes (eg. Electronic codebook (ECB), Cipher block chaining (CBC)) [11]. Authenticated Encryption with associated data (AEAD) provides data's confidentiality, integrity and authenticity simultaneously. Galois/Counter Mode (GCM) has been standardized as the authenticated encryption modes [37].

2.4.1 Galois/Counter Mode (GCM)

GCM provides data confidentiality with a symmetric key block cipher of a 128-bit block size, e.g. AES. Also, GCM provides authenticity of encrypted data with a universal hash function defined over binary Galois field. In addition, GCM can provide authentication for additional data without encryption.

The input of GCM includes:

· P : plaintext

· K : secret key

· A : additional authenticated data

· IV : initialization vector

The output of GCM includes:

· C : ciphertext

· T : authentication tag

The authenticated encryption operation of GCM is as follows [1]:

· $H = E(K, 0^{128})$

$$\cdot Y_0 = \begin{cases} IV||0^{31}1, & \text{if len(IV)} = 96 \\ GHASH(H, \{\}, IV) & \text{otherwise} \end{cases}$$

· $Y_i = incr(Y_{i-1})$ for $i = 1, ..., n$

· $C_i = P_i \oplus E(K, Y_i)$ for $i = 1, ..., n$

· ${C_n}^* = {P_n}^* \oplus MSB_u(E(K, Y_n))$

· $T = MSB_t(GHASH(H, A, C) \oplus E(K, Y_0))$

where GHASH(H, A, C) = X_{m+n+1} is defined as:

$$X_i = \begin{cases} 0, & \text{for i = 0} \\ (X_{i-1} \oplus A_i) \cdot H & \text{for i = 1, ..., m-1} \\ (X_{m-1} \oplus ({A_m}^* || 0^{128-v})) \cdot H & \text{for i = m} \\ (X_{i-1} \oplus C_i) \cdot H & \text{for i = m+1, ..., m+n-1} \\ (X_{m+n-1} \oplus ({C_m}^* || 0^{128-u})) \cdot H & \text{for i = m+n} \\ (X_{m+n} \oplus (len(A) || len(C))) \cdot H & \text{for i = m+n+1} \end{cases}$$

2.5 Keccak Hash Function

The Keccak hash function is the winner of SHA-3 selection [56] and it uses a sponge construction. For hashing, the message is absorbed every 1600 bits, and the Keccak-f[1600] is the speed-critical part.

The Keccak-f permutation function goes through 24 iterations of the following 5 steps on 1600 bits of data, which are treated as a block of 5×5 64-bit words:

· θ **step:** XOR the 5 lanes of each column.

· ρ **step:** Rotate lanes with a defined offset.

· π **step:** Transpose lanes in a fixed pattern.

· χ **step:** Non-linearly combine AND gates and INV gates for each row and then XOR with the rwo.

· ι **step:** XOR a single lane with a round constant

2.6 Existing Secure ICs

CryptoMemory

Atmel's Cryptomemory [8] is the first '*secure memories with authentication*' [7], which appeared in 1999. This device offers cost-efficient, high-security electrically erasable programmable memory chips (EEPROMs) and host-side security for applications requiring comprehensive data protection.

The advantage of such kind of IPs is their security features, low cost and ease of deployment. They can serve as an add-on item to exiting platforms, providng a safe place for sensitive data. There are plenty of applications of these devices, for example, AT88SCxxxxC series are used to store High Bandwidth Digital Content Protection (HDCP) keys in products such as NVidia grahpic cards [58].

However, the authentication protocol inside this chip is simple, and is shown to be vulnerable in some academia papers. [28] demonstrates that the secret key can be recovered in 2 to 6 days, using 200 CPU cores. [75] improves the attack by using ASIC, and the key can be retrieved in 0.55 days. Also, there are flaws on the physical implementations. [9] discovers that Atmel CryptoMemory doesn't protect against power analysis attacks. The author enables 'RST' signal to disable counter increment, then they are able to recover the 64-bit authentication key with 100 power traces in 20 minutes.

Authenticators

Another type of secure IC product is Authenticator, which is provided by plenty of companies, such as Atmel, NXP and Maxim. Authenticators are for IP protections, by preventing illegal copies of products as counterfeit protection. These devices are usually low cost by supporting one specific algorithm, and can be easily integrated for authentication solutions into end products.

The functionality of Authentication is that host system giving challenges and sensors sending back response calculated with pre-stored secrets. Only 'good' sensors can calculate the correct response, the host system will validate the result (shown in Figure 2.4).

The authentication protocol can be categorized into three different methods as shown in Table 2.3. From easy to complicated regarding the computation and applications, they are

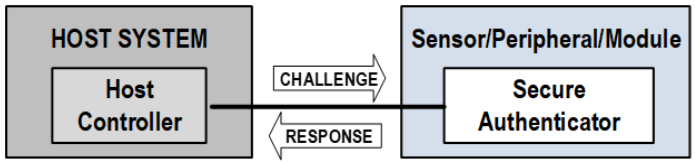


Figure 2.4: Authenticator WorkScheme

Table 2.3: Authenticator Summary

Authenticators	Symmetric		Asymmetric
	SHA-based	AES-based	ECC-based
Functions	Authentication	Authentication/Encryption	Authentication and key exchange
Applications	Clone prevention Home automation	Industrial control	Industrial networking
Product	Atmel Maxim	Atmel	Atmel (launched 2013) Maxim (2013) NXP (2016 rev2)

SHA-based, AES-based and ECC-based methods.

First, symmetric keys can be used for two-way authentication. With symmetric keys, the host and slave must operate from the same secret key, and the secret has to be protected from disclosure attack on both sides. SHA-based devices have fast authentication, while AES-based devices have authentication plus encryption in a single protocol.

Two disadvantages associated with symmetric key-based systems are: (1) key distribution/management, and (2) the need to protect the secret key inside the host system as well as the slave system.

To address these drawbacks, cryptographic algorithms involving asymmetric keys, like ECDSA, are advantageous because the party that is authenticating the peripheral doesn't have to securely store a secret. Instead, the authenticating party can use a public key that can be distributed freely. Thus, asymmetric algorithms solve both the key distribution problem and the need to secure the key in the host system. However, asymmetric key cryptography is usually of higher cost than symmetric-key based algorithms.

CHAPTER III

A Compact AES Accelerator for Mobile SoCs and IoT

This work [81] presents a voltage-scalable AES accelerator targeting mobile SoCs and IoT devices with ∼50 - 500 Mbps throughput, while achieving best-in-class area and energy efficiency. The proposed accelerator is fully synthesizable and implements 128-bit AES using only 2228 logic gates. By eliminating the ShiftRow and MixColumn registers and replacing data and key storage with latches, area is reduced by 41%. This, along with retiming of a 2-stage Sbox design in native GF$(2^4)^2$ composite-field computation, leads to a 3.38× energy efficiency improvement over a baseline implementation at nomial voltage with four 128-bit registers and 1-cycle GF$(2^4)^2$ Sbox methods. The proposed design achieves 1.3 GHz at 0.9V, peak throughput of 494 Mbps, and peak energy efficiency of 446Gbps/W. Implemented in 40nm CMOS, the accelerator area is only 0.00429mm^2, marking the state-of-art smallest AES accelerator considering technology scaling.

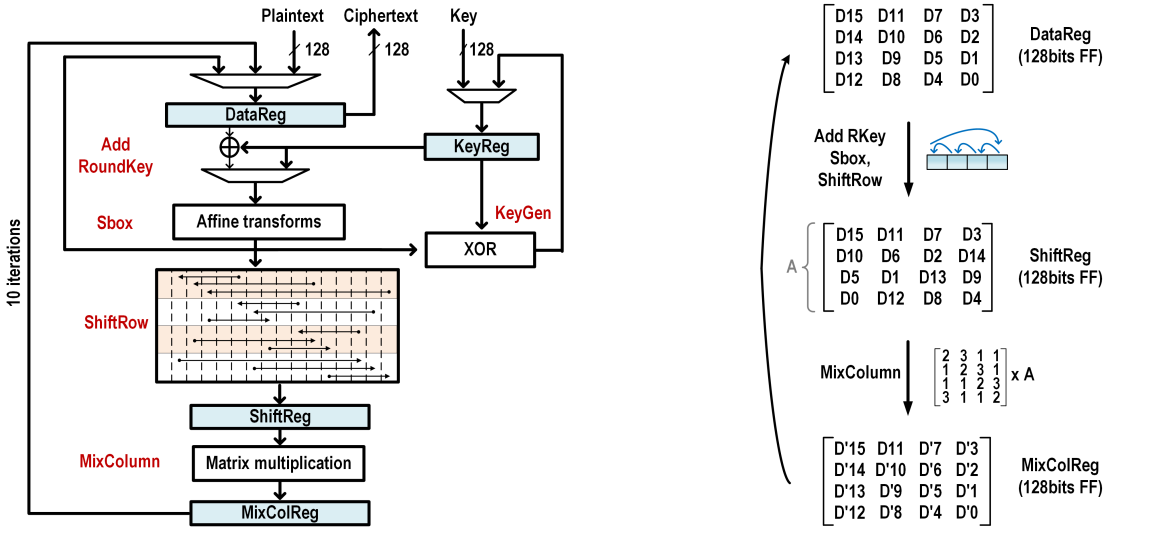
3.1 Previous Work

For mobile devices, silicon area (i.e., cost), throughput, and energy efficiency are all key design constraints. There are several options to perform AES. First we can execute AES on a general purpose processor, however, this is very energy inefficient. For instance, executing AES on an Intel i7 processor requires 7.8 nJ/bit (calculated from [31]).

The 2nd option is to use a standard ASIC implementation. Highly parallelized imple-

mentations [47] provide Gbps throughput, which is critical in server applications. However, their large silicon footprint is disadvantageous in cost-sensitive mobile SoCs. Recently, several energy efficient AES implementations are presented [51, 82]. However, their kbps-range throughput cannot meet the demands of mobile devices with high-speed data streaming. For instance, the ARM Cricket chip [51] uses a standard AES accelerator, however, the energy remains relatively high, at 0.21nJ/bit, which is approximately 300x higher than the energy per bit required for the arithmetic operations in that processor. At the same time, the standard AES accelerator occupies 0.104 mm^2, which is comparable to the size of the processor core in the cricket chip.

3.2 Baseline implementation



(a) Datapath

(b) Data flow described in matrix

Figure 3.1: AES

An AES-128bit is implemented as a baseline comparison, whose datapath is shown in Figure 3.1a. 128bits of plaintext goes through 4 stages of AddRoundKey, Sbox, ShiftRow and MixColumn for 10 iterations. It also requires 128bits of key as inputs, which goes through key generation logic to provide each iteration's key for encryption. The output

of AES is a 128bit ciphertext. Figure 3.1b illustrates the operations on the 128-bit vector represented by the 4-by-4 matrix.

3.3 Proposed Energy Efficient AES

3.3.1 DataReg & ShiftRow

3.3.1.1 Previous Work

A number of efforts have been make in prior work to optimize the DataReg & ShiftRow steps. For example, the Intel work [46] suggests eliminating the ShiftRow register. Instead it achieves the ShiftRow operation when the data is copied back from MixColReg to DataReg, by permutating the data by hardwiring as it is being copied (Figure 3.2). For instance, instead of storing the first column with data byte 15, 14, 13 and 12 which is the standard way, the new data register will store data byte 15, 10, 5 and 0, which is the location after the shiftRow operation. So the technique used in this paper saves the hardware of shift register, which reduces the register count in datapath by 33% and is quite effective.

3.3.1.2 Proposed Datapath

Building upon the above shiftRow technique in [46], our goal is to further reduce the register count by also eliminating the MixColReg.

The idea is shown in Figure 3.3. In each iteration, the data is read directly out of dataReg and stored back in the same register. However, since the location from which the data is retrieved and stored differ, there are cases that the original data hasn't been retrieved yet before the new data is stored, creating a conflict. There are 6 cases of this confict, as highlighted in the matrix. So these 6 bytes need to be delayed and we temporarily store them in the 48-bits storage register. This further reduces the overall register count by additional 21%.

In addition, we implement the dataReg and StorageReg with latches (Figure 3.3), by

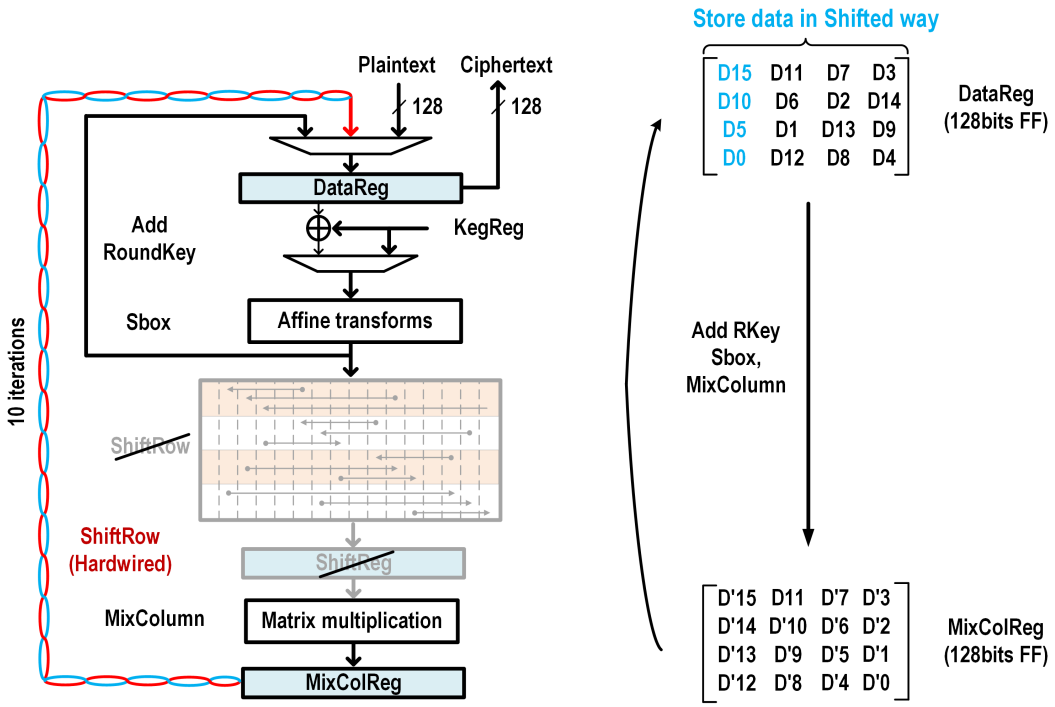


Figure 3.2: Optimized ShiftRow [46]

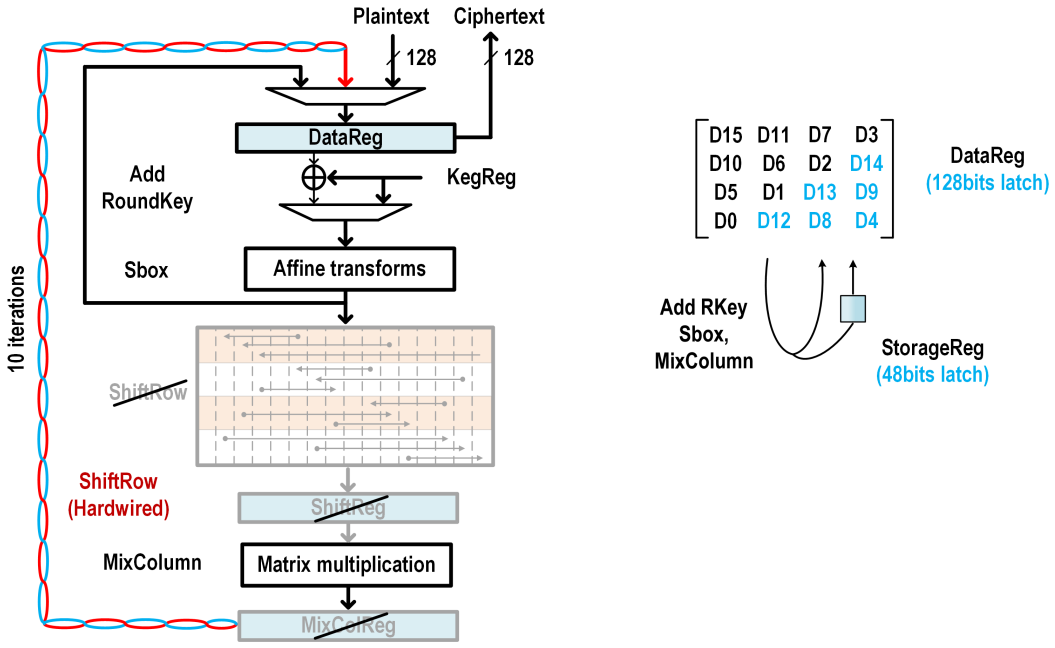


Figure 3.3: Proposed DataReg & ShiftRow Technique

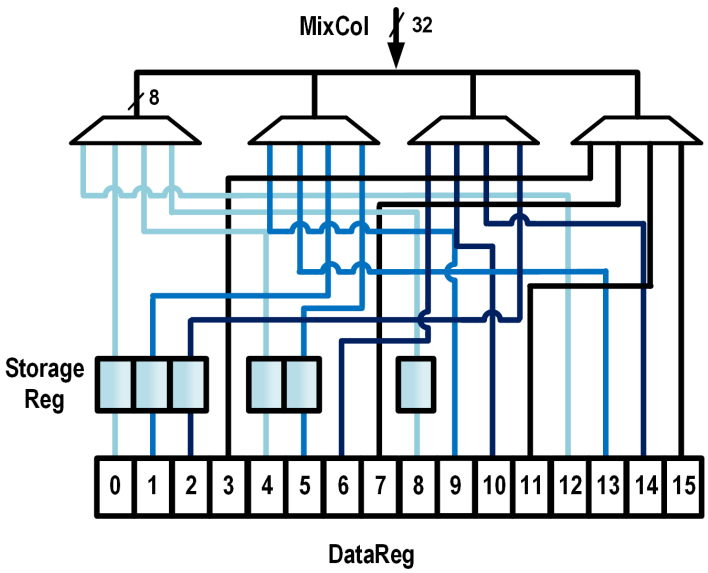


Figure 3.4: Detailed hardwired solution

retiming the path. Because the critical path is not within this stage, it's easy to achieve this retiming during synthesis, place and route steps. This way, we can further save area and power.

The detailed implementation of the hardwired part is shown in Figure 3.4. The 32bits output from mixcolumn state comes in, and goes through 4 muxes and wires, and is finally stored in the correct byte locations in the dataReg. Six byte locations need storage register to hold the new data has been retrieved.

To summarize the proposed architecture, we remove the shiftRow registers and add 48bits storage register for intermediate values. In addition, latches are used as these storage elements.

3.3.1.3 Results

Table 3.1 lists the simulated energy and area results of the three methods discussed before. The proposed approach has a 2.7× energy improvement over a conventional design and has 1.8× energy improvement over [46] work for these three registers. So this table indicates the proposed design is energy efficient and compact.

Table 3.1: Comparison table of DataReg & ShiftRow

	Energy (pJ/cycle)	Area (μm^2)
Conventional	2.98	1355
[46]	1.99	903
Proposed [81]	1.12	466

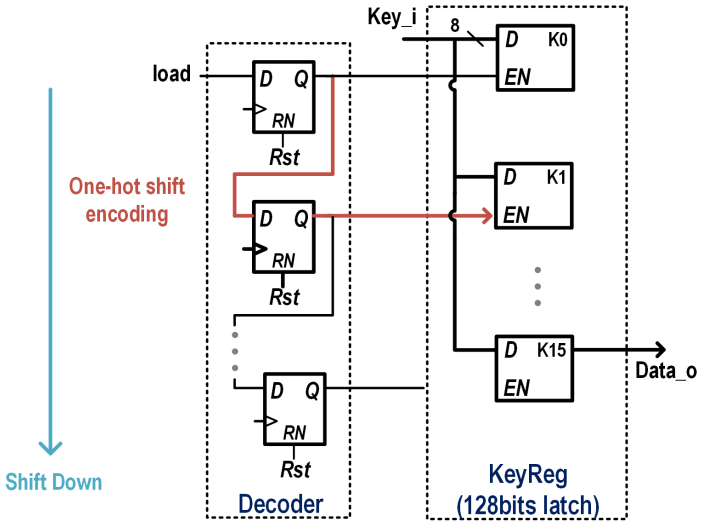


Figure 3.5: One-hot shift encoding

3.3.2 KeyReg

[46] implements the key generation step by shifting all the data down by 1 byte, where the keyReg works as a FIFO, which consumes a lot of power.

Instead of shifting all the 128bits data, our implementation shifts in the decoder part. One-hot shift-based addressing is used for the decoder (Figure 3.5). It uses a cyclic address generator with a single chain of 16 registers, similar to [39]. This way, we reduces the shift power by 80% in the keyReg. In addition, KeyReg is changed from a 128-bit flip-flop register to a 128-bit latch register to further save power and area.

3.3.3 Sbox

3.3.3.1 Previous Work

The 8-bit inversion defined in Sbox is complex and has been the focus of a number of optimization methods. The straightforward approach to implement Sbox is to have a 256-byte SRAM to store precomputed results, since the Sbox operation is just a map of 8 bit input data to determine an 8 bit output data. The 2^{nd} approach is to synthesize this map directly using logic. The 3^{rd} way involves some math calculations called composite field. The data in Sbox stage is in GF(2^8) , and it has an isomorphism transformation to GF((2^4)^2). So we first do the isomorphism to GF((2^4)^2), then do inverse and affine operations, and finally transform back to GF(2^8). The reason to do this field transformation, is that inverse in GF(2^4) is much easier to implement than GF(2^8), so we can obtain an area reduction using this composite field approach.

The simulated results of area, power and cycle time for these three Sbox approaches are shown in Table 3.2. The composite field method achieves the smallest area but has relatively high power and longer cycle time. Therefore, we look into the critical path of this composite field approach to dig into this longer cycle time and higher power.

3.3.3.2 Proposed Technique

As shown in Figure 3.6, the 8-bits data in GF(2^8) comes in and is mapped to GF((2^4)^2), then the two 4-bits number goes through squaring, multiplication, addition and inverse. The two AND gates before the inverse map have three 4-bits data.

We make the observation that, 2 of these paths are fast, while 1 is a long path. The difference in signal arrival time of fast and slow paths will result in dynamic glitches and cause high power. Therefore, we re-time the Sbox datapath by adding 12 flip-flops before the datapath re-converges (Figure 3.6). This equalizes the path delays, therefore saves the power. In addition, the cycle time is reduced by efficiently inserting a pipeline stage.

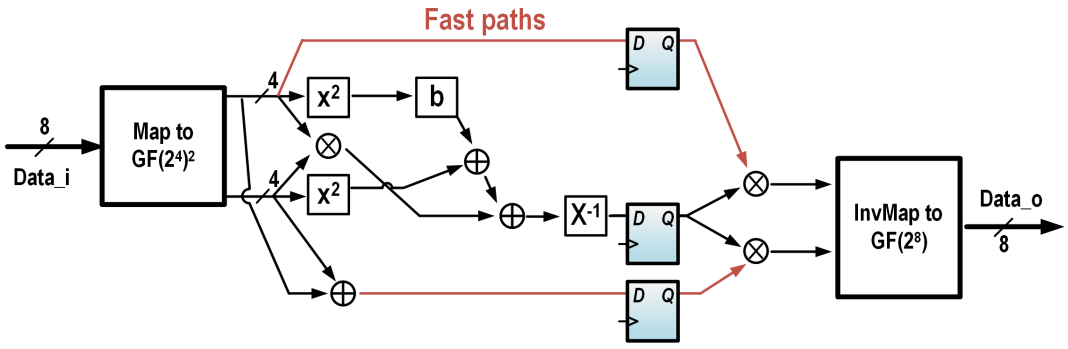


Figure 3.6: Proposed Sbox datapath in GF($(2^4)^2$) with 2 cycles

Table 3.2: Comparison table of different Sbox implementations

S-box Architecture	Area (μm^2)	Power (mW)	Cycle Time (ns)
Look-up table (SRAM)	2175	1.7	0.55
Look-up table (Logic)	816	1.15	0.4
GF$(2^4)^2$ in 1 cycle	558	1.42	0.64
GF$(2^4)^2$ in 2 cycles	582	0.9	0.46

3.3.3.3 Results

As shown in Table 3.2, the proposed method incurs a modest 4.3% area overhead while providing power savings of 37% over the 1-cycle GF($(2^4)^2$) implementation. Also, splitting Sbox into two cycles shortens the critical path, decreasing clock cycle time by 28%.

Since the stages are pipelined in the AES accelerator, splitting Sbox into 2 cycles only increases the total number of cycles by 1, which is less than 0.3% out of the total 336 cycles in the baseline design.

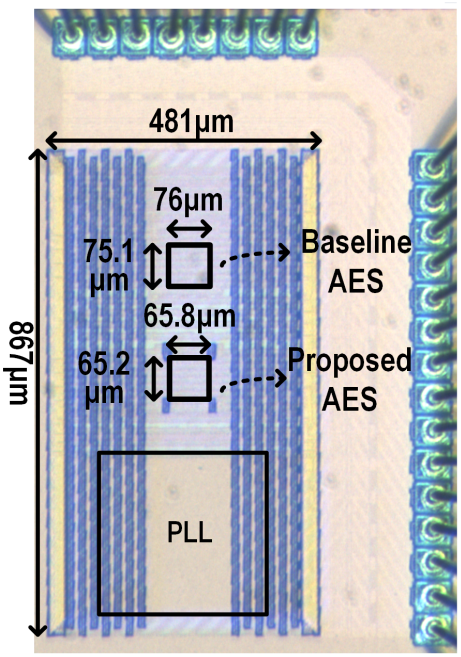


Figure 3.7: AES Chip micrographs

3.4 Testchip and Measurements

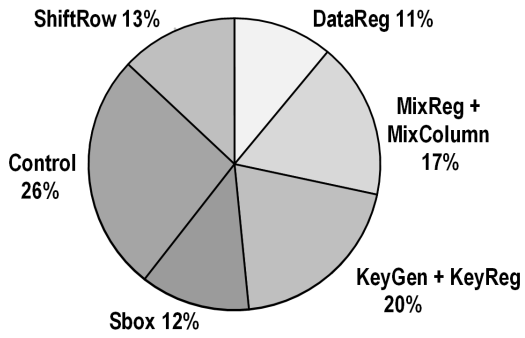
Figure 3.7 is the chip micrograph, where both a baseline AES and a proposed AES design are implemented in 40nm. A phase-locked loop (PLL) IP provides clock on chip.

A baseline is implemented for a fair and clear comparison of each technique. Synthesized in 40nm, it can achieve a frequency of 1.13 GHz at 0.9V, consumes 8.3 mW with area of 0.0057 mm^2, using about 3000 gates (Figure 3.8a). Figure 3.8b shows the simulated power breakdown across different units, showing a relatively balanced design.

The proposed AES accelerator used three lightweight technique, with two for register optimization and one for logic power reduction. At 0.9 V, the proposed design consumes 4.39 mW power and achieves 1.3 GHz frequency with only 0.00429 mm^2 area. Figure 3.9 and Figure 3.10 show the performance and energy measurements of the baseline and proposed AES. The proposed design is fully synthesized, enabling operation across a wide voltage range. At 1V, performance of 1.47 GHz is obtained while peak energy efficiency of 446 Gbps/W is achieved at 0.47V.

Measure Baseline Design 0.9V, 40nm	
Freq.	1.13 GHz
Power	8.3 mW
Area	0.0057 mm^2
Gates	2964

(a) Chip Summary



(b) Baseline power breakdown

Figure 3.8: AES baseline implementation

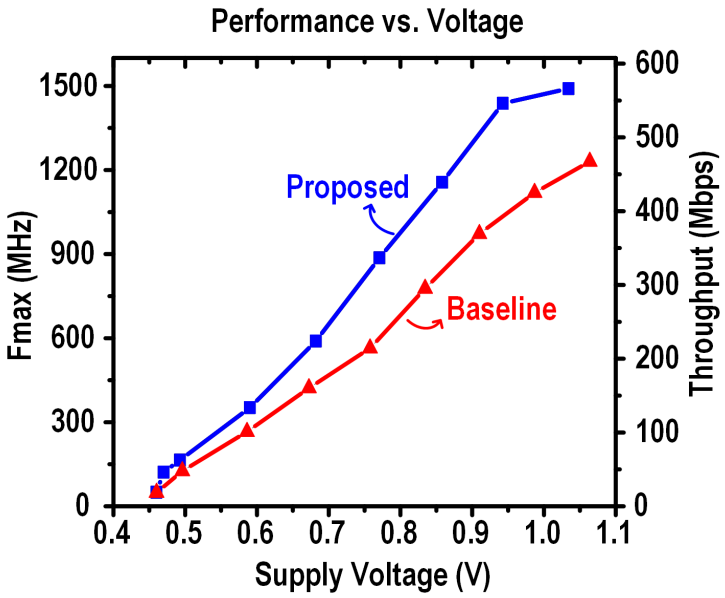


Figure 3.9: Measured performance across voltages of the proposed AES design

Figure 3.11 shows how each of the proposed techniques contribute to power reduction. The total power shown is from measurements, and the relative percentage of each module is from simulation. The first DataReg & ShiftRow technique increases the dataReg power by 20% because of the added storage register. But we see big savings from reducing the MixColReg and ShiftRow operation. The 1-bit shift encoding saves the key generation datapath power by 18%. The 2-cycle Sbox technique saves 33% power in the Sbox stage. Also, the other circuits which includes controls also sees saves 46% power savings. As a result, the total power savings are 47% over baseline design.

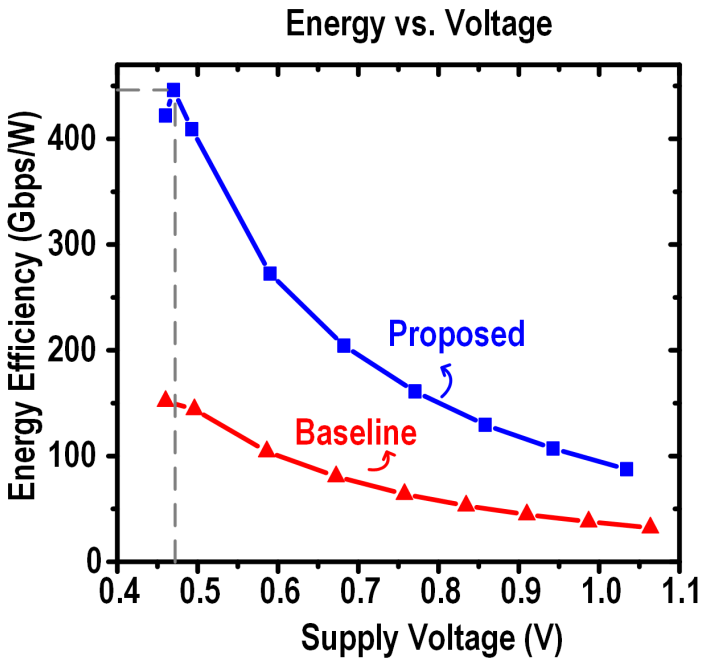


Figure 3.10: Measured energy across voltages of the proposed AES design

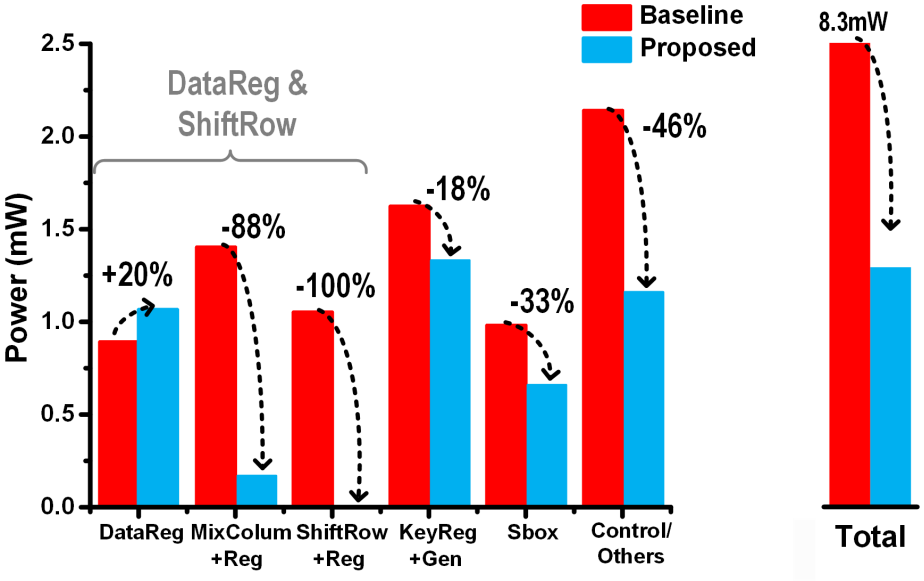


Figure 3.11: Power breakdown of baseline and proposed AES designs

Table 3.3: Chip measurement summary and comparison table of AES designs

	[32]	[82]	[47]	[46]	Proposed [81]
Technology	130nm	130nm	45nm	22nm	40nm
Voltage (V)	Not Reported	0.8 0.75	1.1 0.32	0.9 0.43	0.9 0.47
Power (mW)	17.98 3.9	0.099 0.000692	125 0.409	13 0.45	4.39 0.10
Frequency	290 MHz 130 MHz	12 MHz 100 KHz	2.1 GHz 32 MHz	1.1 GHz 220 MHz	1.3 GHz 122 MHz
Throughput	232 Mbps 104 Mbps	34 Kbps 280 bps	53 Gbps 800 Mbps	432 Mbps 83.6 Mbps	494 Mbps 46.2 Mbps
Energy Efficiency (Gbps/W)	12.9 26.7	0.343 0.405	424 1955	33 186	113 446
Energy/bit (pJ/b)	77.5 37.5	2915 2469	2.36 0.512	31 5.38	8.85 2.24
Number of Gates	3200 3900	5500	Not Reported	1947	2228
Area (mm^2)	Not Reported	<1	0.15 (0.119)	0.0022 (0.0073)	0.00429

Table 3.3 shows the comparison with previous works. For the area comparison, the number in bracket is our calculation to normalize to 40nm for easier comparison, and our design achieves the smallest area considering the technology scaling. For the number of gates comparison, not all paper use the same definition of equivalent gate counts, hence it is hard to give a consistent comparison. The technique used in [46] about optimizing logic for composite field is quite effective, and can also be applied to our design, which could further save the area. Regarding the energy efficiency, we achieve the highest energy efficiency of 446 Gbps/W at 0.47 V among compact AES implementations. The Intel design [47] has a 128bit wide datapath and is optimized for high performance, not compact design, hence it achieves high energy efficiency, but was more than an order of magnitude larger in area.

3.5 Conclusion

To conclude this chapter, security is critical for modern electronic devices in mobile and IoT applications. AES is a widely-used block cipher algorithm for symmetric encryption in a large range of applications. Silicon area, throughput, and energy are all key design constraints. The proposed design achieves high energy efficiency with only 2228 gates. Also, the proposed AES design runs at 1.47 GHz at 1.0V, which satisfies the high throughput requirement for mobile devices.

CHAPTER IV

Recryptor: A Reconfigurable Cryptographic Cortex-M0 Processor with In-Memory and Near-Memory Computing

Providing security for the Internet of Things (IoT) is increasingly important, but supporting many different cryptographic algorithms and standards within the physical constraints of IoT devices is highly challenging. Software implementations are inefficient due to the high bit width cryptographic operations, domain-specific accelerators are often inflexible, and reconfigurable crypto-processors generally have large area and power overhead.

This chapter proposes Recryptor [78, 79], a reconfigurable cryptographic processor that augments the existing memory of a commercial general-purpose processor with compute capabilities. It supports in-memory bit-line computing by using a 10-transistor based bit-cell to support different bitwise operations up to 512-bits wide. Custom-designed shifter, rotator, and Sbox modules sit near the memory, providing high throughput near-memory computing capabilities. We demonstrate Recryptor's programmability by implementing the cryptographic primitives of various public/secret key cryptography and hash functions. Recryptor runs at 28.8 MHz in 0.7 V, achieving 6.8× average speedup and 12.8× average energy improvements over state-of-the-art software and hardware-accelerated implementations with only 0.128 mm^2 area overhead in 40nm CMOS.

4.1 Motivation and Previous Works

IoT platforms have limited computational resources for energy/area reasons. Cryptographic functions typically require high bit-width calculations (64-512 bits), but embedded processors' datapaths tend to be 32-bit wide. Executing these crypto algorithms in software on microcontrollers is simple but energy inefficient and slow. The first option to address this is to optimize software for cryptographic calculations on microcontrollers. For example, [6, 20] propose efficient assembly implementations by manipulating the data flow to maximize the register use, achieving around two to three orders of magnitude of speedup. A second option is to use application specific integrated circuits (ASICs), or accelerators, to run a specific algorithm. ASICs [61, 73, 81] achieve high performance with low energy consumption, but they are only useful for a single purpose, so multiple ASICs are required to cover a range of applications. Even for the same function, there are many different designs optimized for various performance and power requirements [46, 47]. Finally, a third option is to build cryptographic coprocessors for supporting different algorithms, which can give higher throughput and maintain flexibility. The key idea in coprocessor designs is that they try to share as much logic as possible among the supporting algorithms to save area; nonetheless, they still tend to have high area and power overhead since they implement an entire separate processor with fetch, decode, register file and local memory [35, 43, 64]. Also, existing co-processor designs only cover a limited range of cryptographic algorithm types; for example, [35] supports only symmetric and asymmetric crypto algorithms, while [64] supports only symmetric and hash functions.

The importance of security and the limitations in existing solutions motivates a new architecture in favor of the requirements for IoT devices. In this paper, we propose a reconfigurable cryptographic processor, called Recryptor [79], which exploits in-memory and near-memory computing to achieve high energy efficiency, performance, and programmability for IoT security applications. We measure Recryptor's speed-up and energy gains on core functions for symmetric and asymmetric cryptography as well as hash functions. Compared

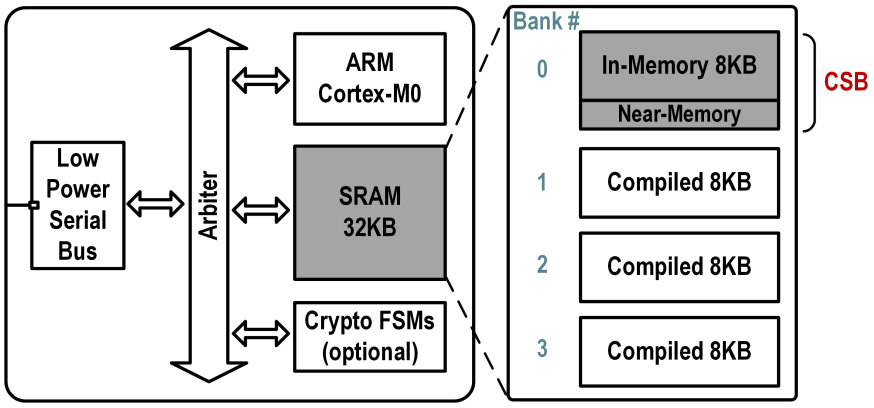


Figure 4.1: Proposed Recryptor architecture.

with a Cortex-M0 baseline, we achieve energy gains of 9.1× for AES, > 6.7× for elliptic curve cryptography (ECC) finite field multiplication and reduction (FFMR) and 4.9× for SHA-3 Keccak function, with energy gains of > 4.1× across crypto algorithms relative to the literature. To the best of our knowledge, this is the first work that can accelerate public/secret key cryptography and hash functions at the same time.

4.2 Proposed Recryptor Overview

Recryptor contains a standard ARM Cortex-M0 microcontroller with 32KB memory, a low power serial bus to access off-chip data, and an arbiter as its internal bus, as shown in Figure 4.1. The optional finite state machines (FSMs) for further acceleration will be discussed in Section 4.5. The 32KB memory is composed of four 8KB banks. Three are implemented using a standard memory compiler while the fourth is a custom designed Crypto-SRAM Bank (CSB).

The CSB can operate as a normal memory with 32-bit read and write, but it also supports large bitwidth in-memory and near-memory computing. As shown in Figure 4.2, the CSB uses a 10T bitcell, which has dual read ports in order to enable different bitwise operations. Each cycle, two words are accessed simultaneously in the bank and perform a bit-wise logic operation which is read out with standard sense amps. The memory sub-banks are set to

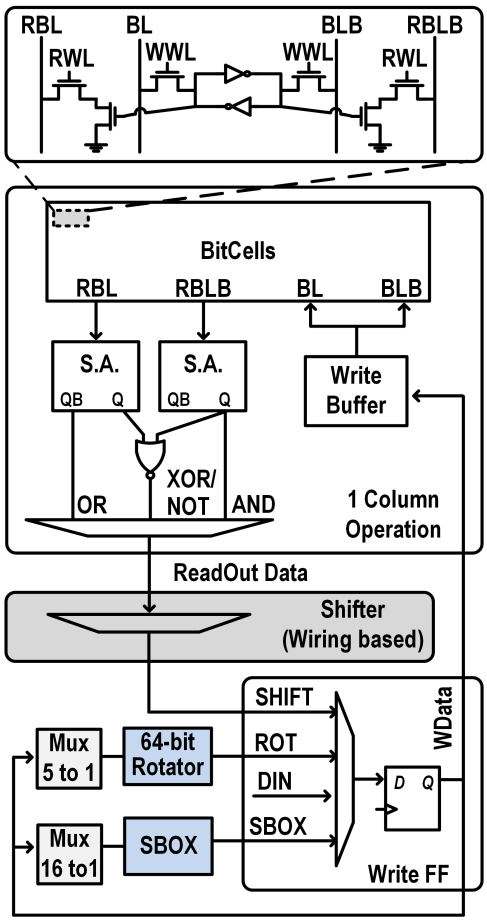


Figure 4.2: Proposed Crypto-SRAM Bank (CSB).

different widths to support different lengths of vector computation. After the sense-amps 'ReadOut Data', there are three near-memory logic functions: a shifter, an arbitrary 64-bit rotator, an SBOX. These three options, together with DIN from the arbiter interfacing with the processor, provide the write-back data to memory.

Equipped with in-memory and near-memory computing capabilities, users can directly program the Cortex-M0 to use the CSB through a memory-mapped decoder to accelerate different crypto algorithms. We optimize and demonstrate AES, FFMR with four different word lengths, and the Keccak hash function in this design.

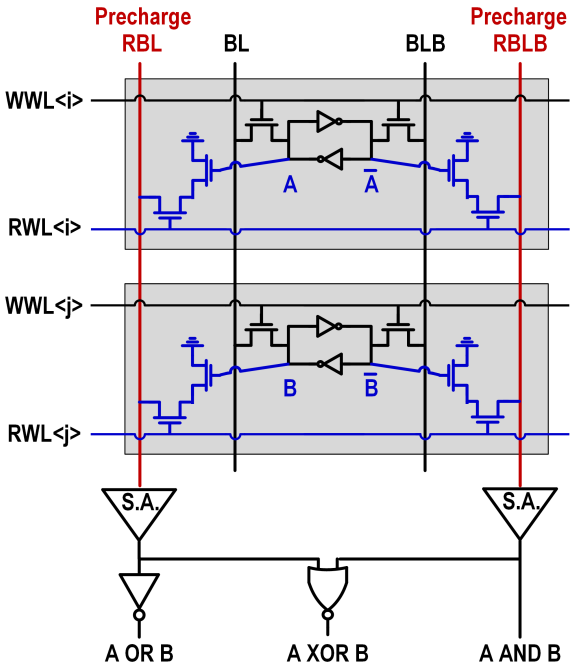


Figure 4.3: 10T bitcell and supported bitwise operations on two words.

4.3 Recryptor's In-Memory Computing

4.3.1 10T Bitcell and Accelerated Bitwise Operations

The 10-transistor bitcell is used in the CSB (Figure 4.2). For normal read operation, only read bitline (RBL) is precharged to high. RBL will be discharged if the stored data is 1 and remain high if the stored data is 0. The data is read out by a skewed inverter-based sense-amp. For in-memory computing, either or both RBL and read bitline bar (RBLB) will be pre-charged, depending on the required bitwise operations. Figure 4.3 shows about how to get the bitwise operations on two words. First, A OR B is achieved by pre-charging RBL to high and then enabling two words' RWL. If at least one of the stored data A or B is 1, RBL will be discharged, which is the function of A NOR B. Similarly, A AND B works the same way, but operates on RBLB. Lastly, A XOR B is achieved by pre-charging both RBL and RBLB, and adding a NOR gate between them.

The supported bitwise operations of CSB are OR/AND/XOR on two words and Copy/NOT on one word. All of these operations are performed in one memory cycle. Like an 8T bitcell,

we achieve robust operation at low voltage using decoupled read. Since both RBL and RBLB are needed for different logic operations, we have two read ports leading to 10 transistors in the bit cell. Smaller SRAM cells could be used for smaller area. However, conventional 6T bitcell [38] has degraded read noise margins and worse performance at low voltages, and the 4+2T bitcell [21] requires additional voltage supplies.

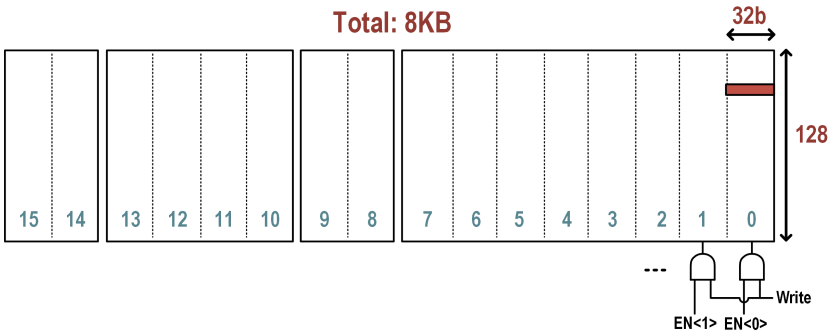
4.3.2 The Configuration of Bank Division

The subbank is configured and implemented as shown in Figure 4.4. The 8KB CSB is comprised of 16 slices, each with 128 32-bit words. During a normal 32-bit memory access, just one slice is activated to save energy. During in-memory computing, by enabling different sub-banks or a combination of them, we can support up to 512-bit single-cycle computations. The size and placement of these sub-banks were optimized at design time to support a wide range of security primitives with efficient signal gating, which will be discussed in Section 4.5.1.

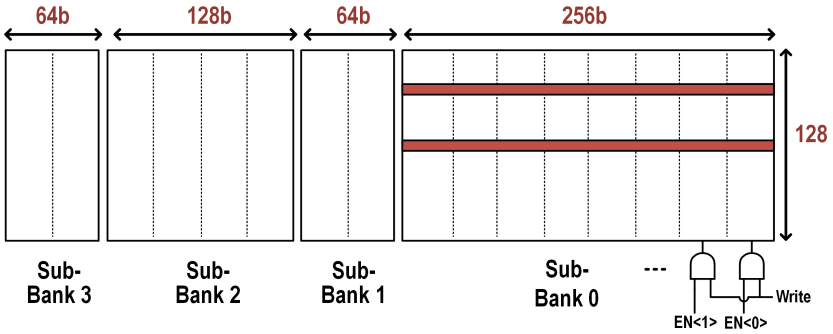
4.4 Recryptor's Near-Memory Computing

4.4.1 Shifter

The shifter implemented is a compact, wiring-based custom design, which is pitch-matched with the SRAM bitcell. There are different MUX options depending on the supported functionality. Table 4.1 shows the functions implemented under each sub-bank, which are used in Algorithm 5/6/7 in Section 4.5. In the 3-to-1 MUX example in Figure 4.5a: vertical wires contain nearby bits' data and a MUX is used to select one of them as the write back data. Figure 4.5b demonstrates a small sample of the physical implementation to show that it is a wiring intensive design. The top is wire with 4 metal layers for horizontal routing and 1.5 minimum spacing. The routing pattern is consistent by shifting 1 via for nearby bits so the design and layout complexity is low.



(a) Normal 32-bit memory access by enabling 1 slice



(b) In-memory computing by enabling sub-banks.

Figure 4.4: The subbank configuration and implementation.

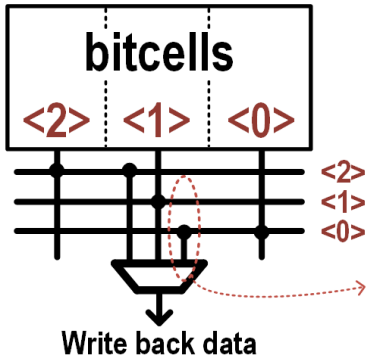
In the Recryptor testchip, the memory read and shifter operations are implemented as a single-cycle operation, before writing back to the write flip-flop as shown in Figure 4.2.

4.4.2 Rotator

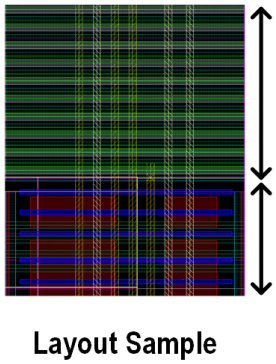
The rotator is a custom 2-stage design for arbitrary 64-bit rotation. As shown in Figure 4.6, the 1st stage rotates 0-7 bits and the 2nd stage rotates in multiples of 8 bits. This architecture is similar as [67], which requires much less area than the long wires needed of directly rotating 0 to 63-bits. We use the same physical implementation, similar as a barrel shifter [62], for both stages in our design. However, transmission gates are used for the muxes to achieve low energy and stable operation at low voltages. By using wire meshes, the same custom compact layout can be used for the first and second stage, which reduces the area of long wires for shifting multiples of 8 bits and design time.

Table 4.1: Shifter Supported Functions

Shifter Options	Function	Supported subBank
LS1	Left shift 1 bit	0,1,2,3
LS4	Left shift 4 bit	0,1,2,3
LS64	Left rotate 64 bits	0,1
RS64	Right rotate 64 bits	0,1
ROT1_w64	Right rotate 1 bit within 64 bits	0,1
ROT8_w64	Right rotate 1 byte within 64 bits	2
S.Row	Shift n_{th} bytes in the i_{th} sub-subBank to $((n+i) \bmod 4)_{th}$ sub-subBank	2
KG	Shift i_{th} sub-subBank words to $((i+2) \bmod 4)_{th}$ sub-subBank	2



(a) Shifter 3-to-1 MUX example



(b) Shifter physical implementation sample

Figure 4.5: Shifter design

4.4.3 Sbox

Sbox is a commonly used component in block ciphers for byte substitution. It is a nonlinear function that performs a multiplicative inversion on GF(2^8), followed by an affine transformation. This algorithm is more efficient with a transformation into the composite field GF(2^4)^2 [46]. Our previous design [81] proposed a 2-stage implementation by adding flip-flops to reduce glitch power due to the presence of fast and slow paths. In this design as shown in Figure 4.7, we further replace the middle flipflops with latches due to a longer clock-cycle, which enables a 1 cycle latency of this block as well as saving a bit of area.

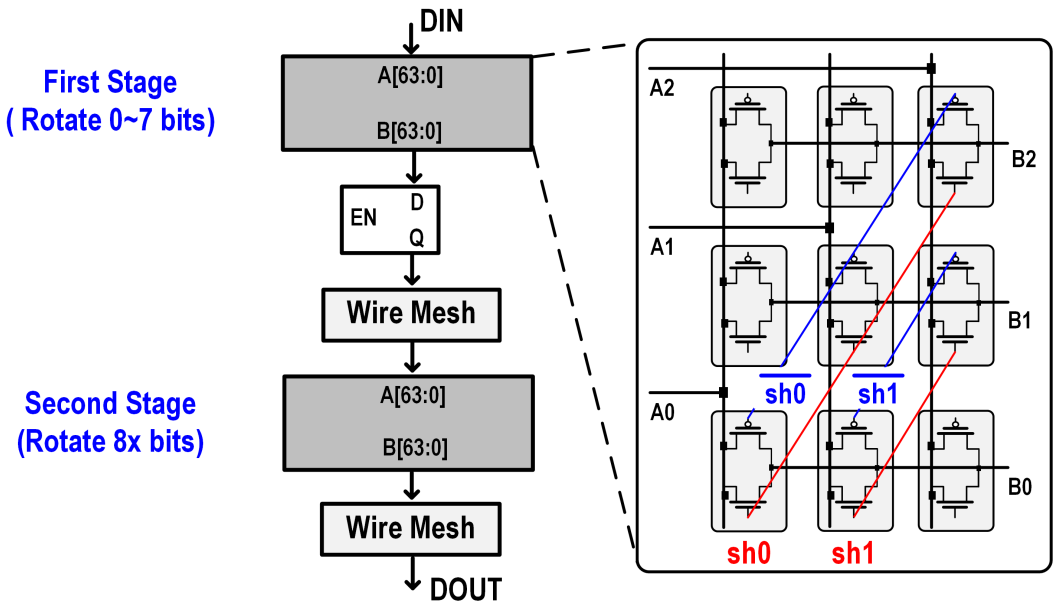


Figure 4.6: Arbitrary 64-bit Rotator.

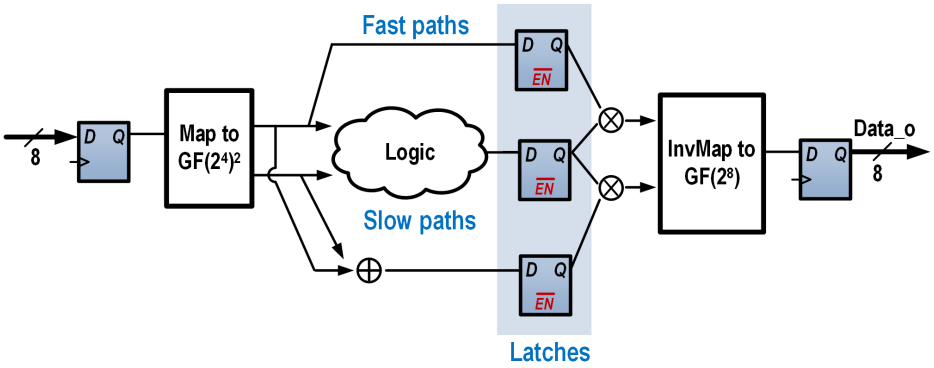


Figure 4.7: Proposed SBOX implementation.

4.5 Programmability and Optimized Algorithm Implementations

Users can write software to configure Recryptor to accelerate different cryptographic algorithms. We demonstrate one algorithm for each category to show Recryptor's flexibility and performance. First, we will analyze Finite Field Multiplication and Reduction (FFMR), the basic operations for ECC. Then, we will analyze AES for secret key cryptography and Keccak-f for hash functions.

4.5.1 Finite Field Multiplication and Reduction

Field multiplication computes $x \cdot y = z$, where x and y are binary polynomials of degree at most $(m-1)$, and z is of degree at most $(2m-2)$. In order to return the multiplication results to degree of at most $(m-1)$, field reduction computes $z \bmod r = c$, where r is a polynomial of degree m.

The L*ó*pez-Dahab (LD) field multiplication algorithm [33] uses a windowing method with a precomputed table. With a window width of ω bits, the input x is shifted right to scan its last ω bits at a time. Then each ω bits are used as an index for the precompute table lookup. However, the number in the finite field needs m bits, and the inputs/output/intermediate values are stored in the memory. Due to register spilling on the M0, using this algorithm tends to create a large memory accesses. [6, 20] optimizes the overflow to solve this spilling problem by maximizing the register reuse.

We propose a new optimization to combine the (LD) field multiplication and reduction algorithm with the goal of reducing the number of operations on Recryptor, as shown in Algorithm 5. The problem of register spilling doesn't need to be considered in this proposed method since all the calculations are computed in /or near memory with m bits of parallelism . Table 4.2 lists the estimated number of operations needed for each step in Algorithm 5, with respect to the curve degree m and window width ω. The first step computes 2^{ω} of m bits numbers; 0·y and 1·y need memory reads with copy and data writeback; and all others need the XOR operation between two words. For 2^{t}·y (t = 0... ω-1), additional SHIFT operations

Table 4.2: Estimated # operations according to Algorithm 1.

Step #	Mem read w/ XOR + Mem Write	Mem read w/ Shift + Mem Write	Mem read w/ Copy + Mem Write	Mem read w/ Shift
1	2^{ω}- 2	ω - 1	2	
2*	2^{ω}- 2	ω - 1	2	
3			1	
5				$\lceil m/\omega - 1 \rceil$
6	$\lceil m/\omega - 1 \rceil$	$\lceil m/\omega - 1 \rceil$		
7*				$\lceil m/\omega - 1 \rceil$
8*	$\lceil m/\omega - 1 \rceil$			

before XOR are needed to remove the overflowing bits. Step 5 would apply SHIFT to the input x and use the first ω bits as the index to the precompute table in Step 1. Then, in Step 6, we do an XOR and a 4 bit SHIFT. Reduction is considered within our algorithm with steps of 2/7/8, in order to reduce the size of intermediate values.

Algorithm 5 López-Dahab multiplication and reduction in $\mathbb{F}_{2^m}$ with windows of width $\omega = 4$ on Recryptor

Input $x = (x_{\text{m-1}}, \ldots, x_0)_2$, $y = (y_{\text{m-1}}, \ldots, y_0)_2$, $r = (r_{\text{m}}, \ldots, y_0)_2$
Output $c = xy \bmod r = (c_{\text{m-1}}, \ldots, c_0)_2$

1: Compute $T(\mu) \leftarrow \mu y \bmod r$ for all polynomials μ of degree at most $\omega - 1$
2: Compute $T'(\mu) \leftarrow \mu r$ for all polynomials μ of degree at most $\omega - 1$
3: $C \leftarrow 0$
4: **for** $j \leftarrow \lceil m/\omega - 1 \rceil$ down to 0 **do**
5: $\quad \mu = (\ \mu_{\omega-1}\ , \ldots,\ \mu_0\ ) = (x >> j \cdot \omega)$ & 0xF
6: $\quad C \leftarrow \text{SHIFT}(C \bigoplus T(\mu), \text{LS4})$ (Syntax: SHIFT(x,y): apply y shifts to x using Shifter)
7: $\quad \mu' = (c >> m)$ & 0xF
8: $\quad c \leftarrow c \bigoplus T'(\mu')$ (Note: reduction step, c stays to be m bits)
9: **end for**
10: **return** c

Table 4.3 shows where to perform calculations of different word lengths by activating different sub-banks. The sub-bank is configured to select nearby banks for the options listed, with efficient signal routing for shift operations. With the estimated number of operations in the column 2/3/4 to be 2, and the last column to be 1, the total number of estimated operations for multiplication in $\mathbb{F}_{2^{233}}$ is 330, compared to 4980 on a baseline Cortex-M0 software implementation and 2968 with register optimization [20].

Table 4.3: Enabled sub-bank of different word length for FFM.

Word length \ Sub Bank	3 (64bits)	2 (128bits)	1 (64bits)	0 (256bits)
163 bits		x	x	
233 bits				x
283 bits			x	x
409 bits		x	x	x

4.5.2 AES

Algorithm 6 shows in detail the implementation of AES on Recryptor. Steps 1/3/4 use 128-bit operations, and step 5 uses the Sbox block on one byte for 16 cycles in total. Steps 7-10 implements the MixColumn operation by left rotating by 1/2/3 bytes, adding intermediate values D and E, and then multiplying based upon the MSB for each byte in H[j]. The KeyGen operation is achieved using 32-bit operations in Steps 11/13/14 and 8-bit operations in Step 12.

Algorithm 6 AES Encryption on Recryptor

Input: plaintext P, key K, where P/K is 128bits and at 1 physical line in bank, $K = [k[0]; k[1]; k[2]; k[3]]$

Output: ciphertext C

1: $C \leftarrow P$
2: **for** $i \leftarrow 0$ to $nr - 1$ **do**
3: AddRoundKey: $C = C \bigoplus K$
4: ShiftRows: $D = \text{SHIFT}(C, \text{SRow})$
5: SubBytes: $D[j] = \text{SBOX}(D[j]),\ j \in [0, 15]$
6: **if** $j \neq nr - 1$ **then**
7: MixColumns: $E = \text{SHIFT}(D, \text{ROT8});\ F = \text{SHIFT}(E, \text{ROT8})$
8: $G = \text{SHIFT}(F, \text{ROT8});\ H = D \bigoplus E$
9: $I[j] = H[j][7]?\ 0\text{x}1\text{B} : 0\text{x}0$
10: $C = H \bigoplus I \bigoplus E \bigoplus F \bigoplus G$
11: KeyGen: $k[4] = \text{SHIFT}(k[3], \text{KG})$
12: $k[4][j] = \text{SBOX}(k[4][j]),\ j \in [0, 3]$
13: $k[4] = k[4] \bigoplus \text{Rcon}[i];\ k[0] = k[0] \bigoplus k[4]$
14: $k[j+1] = k[j+1] \bigoplus \text{SHIFT}\ (k[j], \text{KG}),\ j \in [0, 2]$
15: **end if**
16: **end for**
17: **Return** C

The input/output/intermediate 128-bit values are all stored in the CSB's sub-bank 2, where more shifting options used by AES are supported (see Table 4.1). Compared with an AES ASIC design, our solution only needs an additional Sbox block, which occupies 30% of the area of the current smallest AES ASIC design [81].

4.5.3 Keccak-f

The Keccak-f permutation function [27] with b= 1600, goes through 24 iterations of 5 steps on 1600 bits of data, which are treated as a block of 5×5 64-bit words. Algorithm 7 shows the detailed implementation of the Keccak function on Recryptor. The 320bit-wide operations include the in-memory bitwise operations of AND/NOT/XOR/COPY and the near-memory operations of left/right 64-bit shifts.

Algorithm 7 Keccak-f on Recryptor

Input: Keccak[b](S), where $S' = S[0:4, y]$ is at 1 physical line, $y \in [0, 4]$
Output: S

for $i \leftarrow 0$ to $nr - 1$ **do**
  θ step: $C = S'[0] \bigoplus S'[1] \bigoplus S'[2] \bigoplus S'[3] \bigoplus S'[4]$
  $D =$ SHIFT(C, LS64) $\bigoplus$ SHIFT(SHIFT(C, RS64), ROT1)
  $S'[y] = S'[y] \bigoplus D, y \in [0, 4]$
  ρ step: read $S'[y]$ in 1 cycle, then $S[x, y] =$ ROT($S[x, y], r[x, y]$)
  π' step: $S'[y] =$ **do** SHIFT($S'[y]$, LS64) **for** y iterations
  [Note: π' step result is the transpose of π step in odd iterations]
  χ step: $E[y] =$ SHIFT($S'[y]$, LS64)
  $S'[y] = S'[y] \bigoplus$ (NOT $E[y]$) AND SHIFT($E[y]$, LS64)
  ι step: $S[0, 0] = S[0, 0] \bigoplus RC[i]$
end for
Return S

Keccak is slow to run on a 32-bit processor since it operates on 64-bit words. More specifically, there are two main issues with implementing this function on Recryptor. First, the arbitrary rotation of a 64-bit number in the ρ step would be expensive on 32-bit MCUs like the Cortex-M0. [65] proposes the bit-interleaving technique in software, by collecting even and odd positions of a 64-bit data into two 32-bit data. In contrast, Recryptor applies a custom 2-stage 64-bit rotator placed in the near-memory computation block to save energy

and time on this step. The second problem is that the array transpose in the π step [72] would be very hard to do in-memory since in-memory operations require aligned data. To address this problem, the proposed π' step combines the even/odd iterations and modifies the intermediate results of each iteration, which helps avoid the large memory accesses needed for matrix transpose.

4.5.4 Cryptographic Finite State Machines

Programming Recryptor in software requires only writing a command to a memory-mapped decoder. The command indicates the opcode and which word lines to compute on and write back to. For in-memory operations, input data must be aligned inside memory. This requires operand locality and is addressed in [2].

Since loads and stores on the M0 are somewhat expensive, to further improve performance, we implemented optional FSMs to automatically issue the commands at a small area overhead.

4.6 Testchip and Measurements

4.6.1 Testchips and Measurements

Both the baseline and the Recryptor designs, based upon the ARM Cortex-M0 processor, were implemented on a testchip in 40nm CMOS. Chip micrographs are shown in Figure 4.8. The switch from an 8KB compiled SRAM to a custom compute memory increases its area from 55k μm^2 to 180k μm^2. The layout of CSB is compact, with bitcell banks around the side, memory decoders in between banks, and the shifter, timing generation, and rotator in the center. The area overhead of crypto-FSMs for AES/ECC/Keccak are 0.29k/2.67k/0.62k μm^2, respectively. The total area overhead of Recryptor over the baseline is 36%, including the interface bus.

However, we used design-rule bitcells for the CSB, and if we shrunk them to push-rule

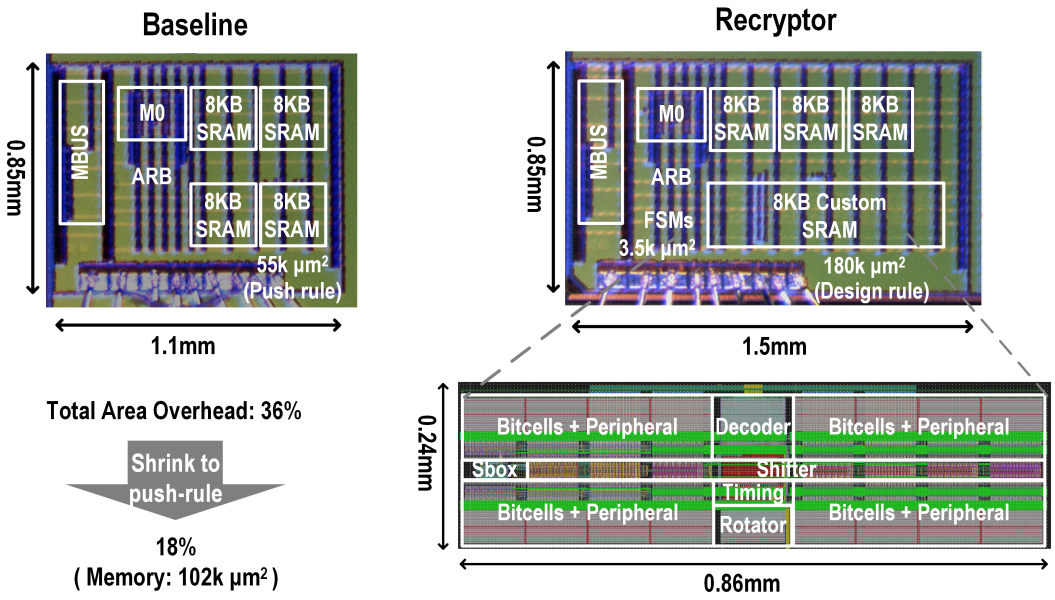


Figure 4.8: Die photo of baseline and Recryptor in 40nm CMOS.

bitcells (with bitcell area shrunk by 50% and row peripherals area shrunk by 30%), the memory could be expected to drop to 102k μm^2. Therefore, area overhead over the existing SRAM is only 47k μm^2. The total area overhead would drop to 18%.

Figure 4.9 shows the measured maximum frequency (Fmax) of the custom 10T SRAM. The blue line shows the Fmax of normal 32-bit reads, while the red line shows the operation sequence of wide in-memory computing reads, near memory shifts, and then data writeback. Since the same sense-amp is used for the normal and in-memory computing reads, the worst case Fmax for them should be the same. At low voltages, the Fmax of in-memory computing is limited by the shifter. Figure 4.10 shows the minimum functional voltage (Vmin) across temperatures for normal CSB reads. The Vmin of 25 °C/125 °C is 0.46V/0.66V, respectively.

Figure 4.11 shows the measured performance of the baseline and Recryptor chips. They both achieve similar frequencies, with a slight difference attributed to die-to-die variation. Recryptor's power is at most 30% larger than the baseline across voltages (Figure 4.12), but this overhead is outweighed by the application speedup. The overall performance improvement and energy reduction of each algorithm will be discussed in the next subsection.

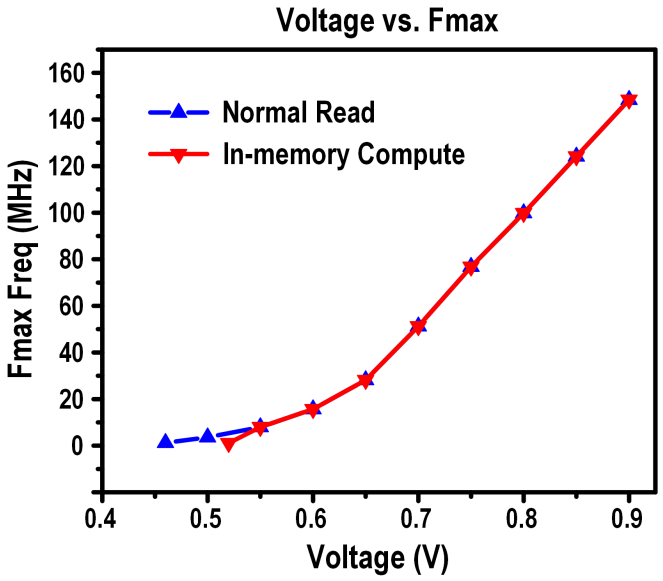


Figure 4.9: Measured Fmax of the custom 10T SRAM.

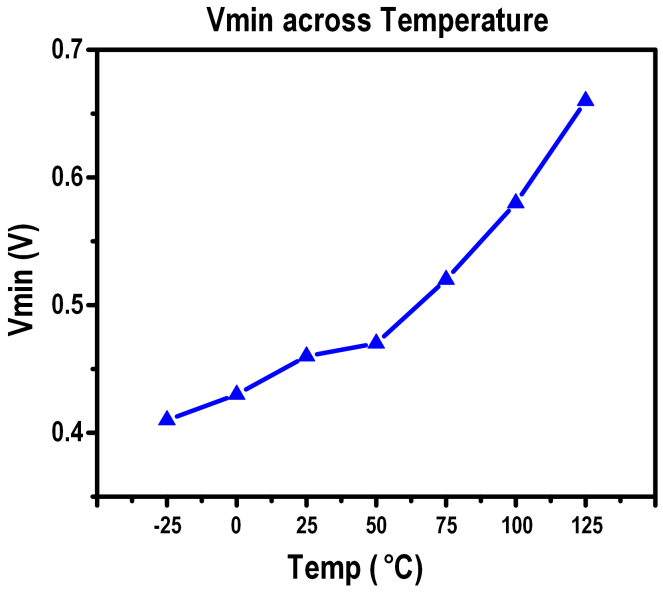


Figure 4.10: Measured Vmin across temperature of the custom 10T SRAM.

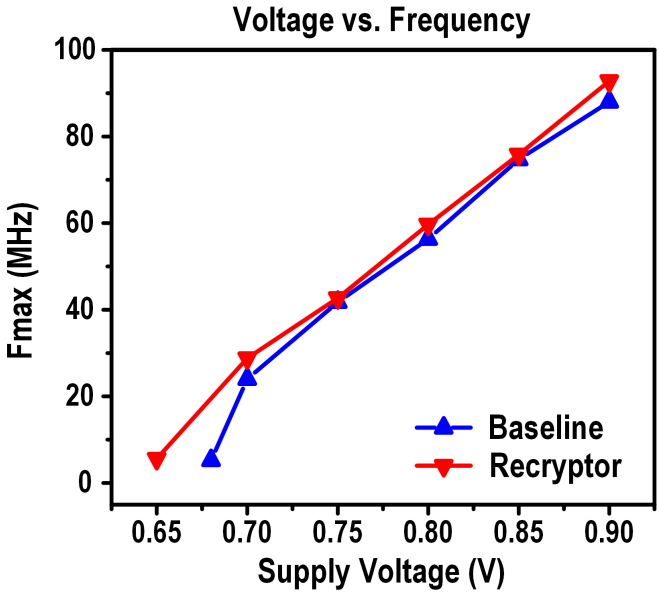


Figure 4.11: Measured frequency of Baseline and Recryptor across voltages.

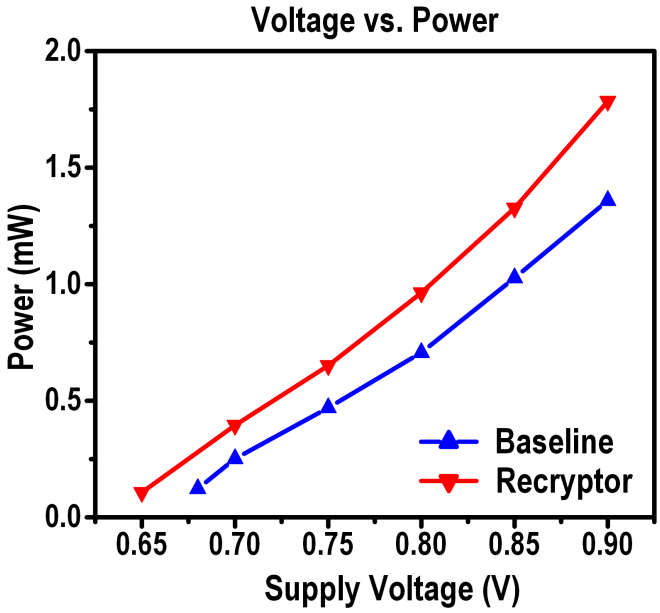


Figure 4.12: Measured power of Baseline and Recryptor across voltages.

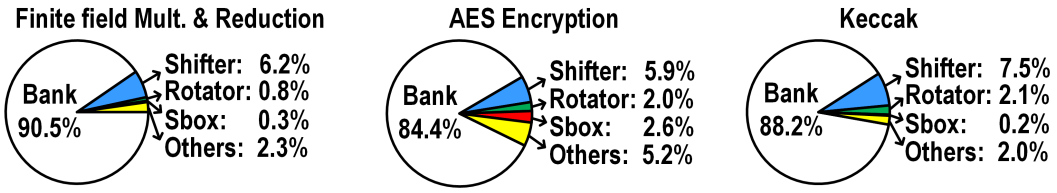


Figure 4.13: Simulated power breakdown of different security functions.

4.6.2 Results Among Algortihms and Comparisons

Different in-memory operations and near-memory logic are activated when running different crypto functions. Figure 4.13 shows the simulated CSB power breakdown. Table 4.4 provides the overall comparison results of three implemented algorithms running on the baseline Cortex-M0, Recryptor, and other processor-based implementations [20, 35, 64].

For Finite Field Multiplication and Reduction, we support word lengths of 163-409 bits. Recryptor achieves $> 11\times$ speedup and $> 6.7\times$ energy savings over the baseline software [6]. The performance improvements and energy reductions increase as the word length increases, showing that Recryptor scales well to large bit-width operations. An assembly-code optimization [20] is included for 233-bit, and this method requires extra effort when the word width changes, which is inefficient regarding the design time.

For Keccak, we use as our baseline the implementation from [50]. To the best of our knowledge, there has not been any co-processor implementation supporting Keccak, so an ASIC design [61] is included in Table 4.4 for comparison.

For AES, we use as our baseline the software implementation from [42]. Compared to this, Recryptor achieves around 9x speedup and energy. Figure 4.14 shows further comparisons in log scale, including an ASIC [81] and a crypto-coprocessor [64] design. Overall, Recryptor serves as an intermediate solution among the compared architectures in terms of area, throughput, energy and programmability.

Table 4.4: Comparison table of different crypto algorithms and designs.

Applications		Designs	#cycles	Freq (MHz)	Time (us)	(Norm.)	Energy (nJ)	(Norm.)
AES		Baseline	6358	24	265	(1x)	64.2	(1x)
		[17]	5429	0.847	6410	(24x)	10259	(160x)
		[18]	20	1000	0.02	(7.5E-5x)	124	(1.93x)
		Recryptor	726	28.8	25.2	**(0.1x)**	7.05	**(0.11x)**
Finite Field Multiplication + Reduction	163 bits	Baseline	5966	24	249	(1x)	62.4	(1x)
		Recryptor	678	28.8	23.5	**(0.09x)**	9.30	**(0.15x)**
	233 bits	Baseline	8921	24	372	(1x)	93.4	(1x)
		[10]	3672	48	76.5	(0.21x)	45.9	(0.49x)
		Recryptor	826	28.8	28.7	**(0.08x)**	11.3	**(0.12x)**
	283 bits	Baseline	10809	24	450	(1x)	113	(1x)
		Recryptor	916	28.8	31.8	**(0.07x)**	12.6	**(0.11x)**
	409 bits	Baseline	19319	24	805	(1x)	202	(1x)
		Recryptor	1246	28.8	43.3	**(0.05x)**	17.1	**(0.08x)**
Keccak		Baseline	23015	24	959	(1x)	238	(1x)
		[12]	15427	100	154	(0.16x)	168	(0.7x)
		Recryptor	3329	28.8	116	**(0.12x)**	48.7	**(0.2x)**

[12,17,18]: Simulation only, no silicon implementation. [17]: 350nm; [18]: 45nm; [12]: 130nm; [10]: No technology given, Cortex M0+ processor; only include mult., no reduction.

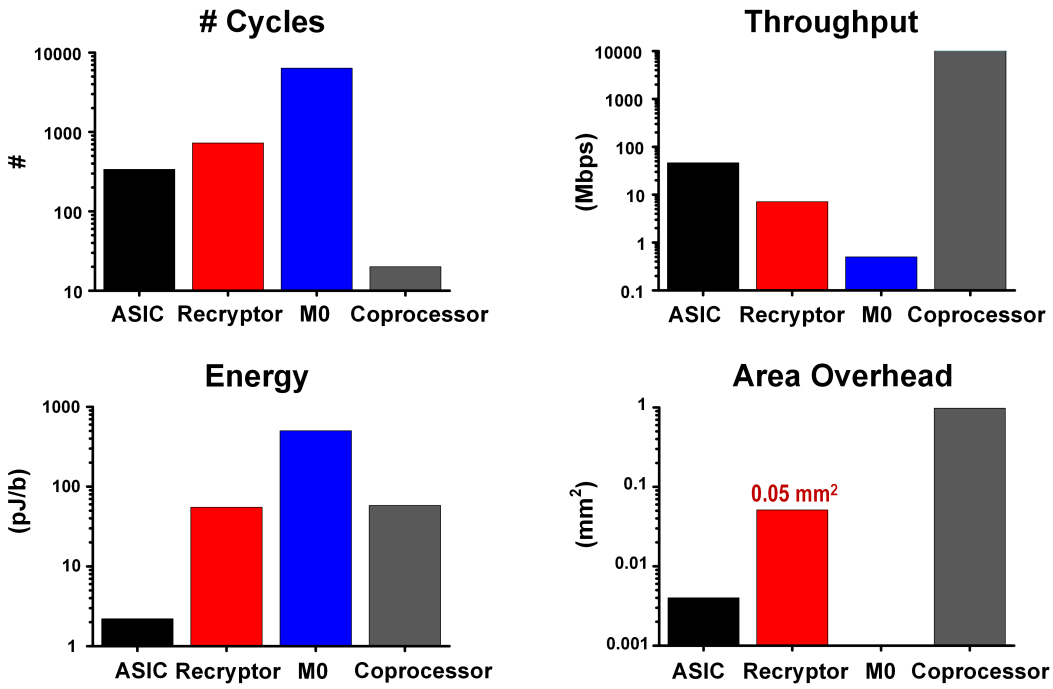


Figure 4.14: Comparison of an ASIC [81], Recryptor, Baseline and a coprocessor [64] design for AES.

4.7 Discussion

4.7.1 Recryptor Optimization

The above performance comparisons among Recryptor, ASICs, and coprocessors depend on the parallelism exploited by the hardware implementations. For example, on AES, Recryptor can perform 128-bit wide XORs, however we used an 8-bit Sbox design with a one cycle delay for area and energy reasons. This creates a performance bottleneck, which explains why the average performance of AES on Recryptor is lower than that of a pipelined ASIC design [81]. However, significantly higher performance can be achieved with a larger Sbox or more shift functions on other sub-banks. On the other hand, on Keccak, our Recryptor implementation achieves throughput similar to that of an ASIC design. This is because Recryptor can exploit 320-bit wide parallelism in Keccak, which is also true of ASIC designs [61].

Table 4.5 shows the minimum required memory capacity for computing and the mod-

Table 4.5: The effective memory capacity for computing and the required modules.

Algorithm	Required Memory Size			Shifter	Rotator	Sbox
	bits/word	# Words	Total Size (bytes)			
AES - 128 bits	128	26	416	x		x
ECC - 163 bits	163	36	734	x		
ECC - 233 bits	233	36	1050	x		
ECC - 283 bits	283	36	1280	x		
ECC - 409 bits	409	36	1840	x		
Keccak	320	20	800	x	x	
All	512	36	1840	x	x	x

ules required for each algorithm, with the last row summarizing all implemented algorithms. Only 1.84KB of memory is required to support the chosen algorithms, although the CSB as implemented here supports 8KB of in-memory computing for greater flexibility. Future optimizations can explore different bank sizes, bitwidths of in-memory operations, and different near-memory modules, depending on the desired algorithms.

4.7.2 Recryptor vs. Near-memory-computing-Only

This section does an ideal calculation and comparison between Recryptor and Near-memory-computing (NMC). Here, NMC refers to the architecture with a normal SRAM bank and near memory logic. The SRAM bank in NMC can read data from a single physical line with N-bits per cycle. The near memory logic in NMC can perform bitwise XOR/AND/OR/INV with up to N-bits per cycle.

In this comparison, both Recryptor and NMC can support algorithms including AES, Elliptic Curve Finite field multiplication (binary elliptic curve up to 409 bits), and Keccak function.

4.7.2.1 Performance

Regarding the long bit-width single cycle operation, followings are the required cycles related with this operation for Recryptor and NMC.

Table 4.6: # Cycles to run Binary Elliptic Curve Finite Field Multiplication and Reduction

#Cycles	**163-bits**	**233-bits**	**283-bits**	**409-bits**
Cortex-M0	7904 (1X)	10152 (1X)	12485 (1X)	19598 (1X)
Recryptor	905 (8.7X)	1157 (8.8X)	1284 (9.7X)	1789 (11X)
NMC (Calculated)	1255 (6.3X)	1604 (6.3X)	1780 (7.0X)	2476 (7.9X)

• **Recryptor** :

· Cycle 1: Read 2 words and Execute bitwise operation

· Cycle 2: Write back

• **NMC** :

· Cycle 1: Read word 1

· Cycle 2: Read word 2

· Cycle 3: Execute bitwise operation

· Cycle 4: Write back

Therefore the calculated number of for NMC is to add 2 times the number of in-memory-computing counts on the Recryptor cycles. The performance comparison between Cortex-M0, Recryptor and NMC on running FFMR is shown in Table 4.6. Here, the Cortex-M0 and Recryptor results are from simulator discussed in the next Chapter, which also includes the cycles for data alignment.

4.7.2.2 Area Overhead

Regarding the area overhead compared with a baseline Cortex-M0 implementation with 8-transistor SRAM bitcell, followings are the required elements for Recryptor and NMC.

• **Recryptor** :

· 10-transistor bitcell

Table 4.7: Calculated Area Overhead of Recryptor and Near-memory-computing (NMC)

(Unit: k μm^2)	**Recryptor**	**NMC**
SRAM-bank	3.1	-
SRAM-peripherals	22.8	-
Near-memory-logic	-	15
Shifter	18.2	18.2
Rotator	3.2	3.2
Sbox	1.3	1.3
Total (k μm^2)	48.6	37.7

  · SRAM peripherals: N pairs of one sense-amp, latch, precharge driver, NOR, and 3-to-1 MUX

  · Shifter / Rotator / Sbox

- **NMC** :

  · Near memory logic: N pairs of XOR, AND, OR, INV, 4-to-1 MUX and buffer

  · Shifter / Rotator / Sbox

  · N bits of flip-flops to store the intermediate word

To calculate the area overhead, we assume Recryptor only supports maximum 409-bits of single cycle bitwise operation, and 1.84 KB of memory supporting in-memory-computation. Also, Recryptor is implemented with push-rule bitcell design. Table 4.7 shows the area overhead comparison between Recryptor and NMC architecture.

4.7.3 Power Analysis Attack

The memory bus is not trusted and is vulnerable to physical attacks, for example, by probing [36]. There have been expensive techniques addressed to solve this problem, for example, oblivious RAM [29] uses address obfuscation and InvisimMem [3] requires 3-D integration to stack DRAM layers on top of logic layers. The Recryptor's idea of compute

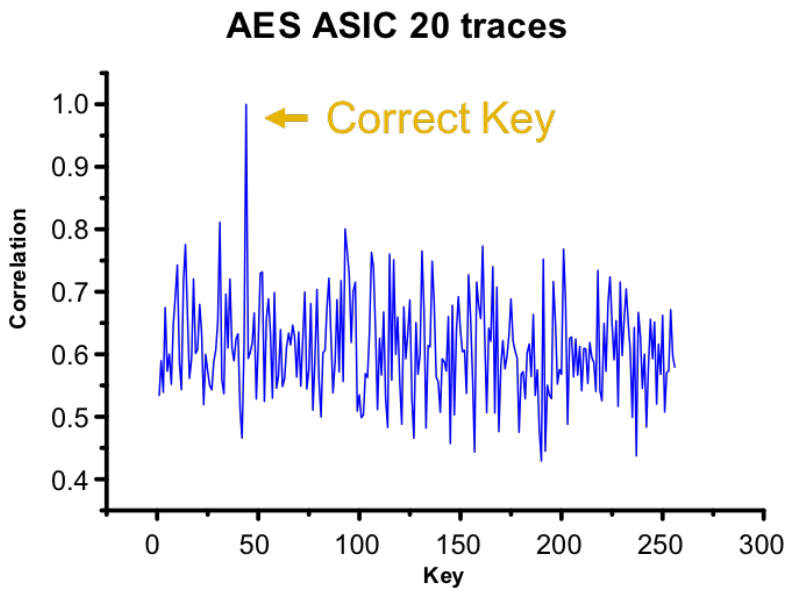


Figure 4.15: Simulated DPA on an AES ASIC with 20 traces.

memories could provide inexpensive encryption to protect the memory bus, taking the attack model to treat in-memory and near-memory as a whole module, but not to probe the internal connection inside CSB.

In addition, a preliminary differential power analysis (DPA) attack is analyzed on Recryptor, for key extractions in AES algorithms. An AES ASIC with 8-bit datapath is also included as a baseline. The attack setup is to first run Hspice simulation on the netlist of the baseline and Recryptor's CSB, and then use MATLAB [59] to calculate correlations and extract every 8-bit key. Each power trace has 20 executions of the 128-bit plaintexts with 1ns sampling time, while keep the same key each time. The DPA result of baseline and Recryptor's CSB is shown in Figure 4.15 and Figure 4.16 . The minimal number of traces to reveal the 1st byte of the correct key (0x2b) is 20 traces for baseline, and 300 traces for Recryptor's CSB. An ideal analysis of DPA attack is shown here, but considerations of real-world problems (e.g. clock jitter) and other attacks (e.g. fault injection attack) are suggested for further analysis.

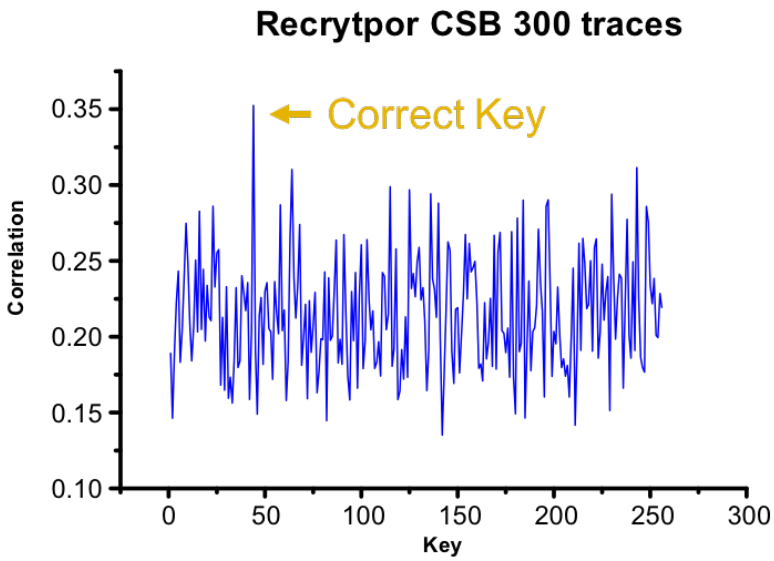


Figure 4.16: Simulated DPA on Recryptor's CSB with 300 traces.

4.8 Conclusion

There are many challenges in IoT security due to the limited computational resources and required flexibility. Current ASICs and coprocessors have limitations in different aspects. In this paper, we proposed a new architecture called Recryptor, which uses in-memory and near-memory computing to efficiently support large vector calculations for crypto algorithms. It maintains programmability and has over 80% runtime and energy savings compared with a baseline processor architecture. Overall, Recryptor is a good intermediate solution in terms of balancing area, energy, throughput and programmability.

CHAPTER V

Recryptor M-ulator: A Simulator for In-Memory Computation

5.1 Motivation and Background

Hardware acceleration has been shown to enable efficient, high performance computation by optimizing datapaths and exploring new architectures. However, building RTL implementations is a time consuming and error prone process. Not only does the generation, verification and synthesis of RTL take days to weeks, but simulating large workloads using RTL is extremely slow and hard to debug. Thus, there exists a need for higher level accelerator design flows which enable rapid design space exploration of new customized architectures.

For many years, computer architects have relied on simulation in order to perform these design space explorations on large workloads. For example, the gem5 simulator [12] supports most commercial ISAs (ARM, ALPHA, MIPS, Power, SPARC and x86), simulates at a high level with reasonable accuracy, and is among the most widely used simulators in the community. The ARM Keil simulator [44] supports ARM7, ARM9 and Cortex-M CPUs with relatively detailed cycle-level pipeline models. However, gem5 does not support microcontrollers, and Keil is not open source, making it difficult to model specialized hardware connected to such CPUs, which represent the types of IoT platforms this dissertation focuses on.

In this chapter, we built Recryptor M-ulator, which extends the M-ulator simulation framework [60] with support for Recryptor functions, to enable simulation of larger workloads with much higher speed than RTL simulations. We use Recryptor M-ulator to evaluate the performance of large workloads like elliptic curve arithmetic and AES-GCM, which may require more memory and/or accelerated hardware functions (such as a right-shifter) that the test chip does not have.

5.2 Proposed Framework

The proposed simulator is based upon the M-ulator project [60], an open source simulator for ARM Cortex M0 and M3 targets. We implemented Recryptor functions by augmenting the memory map to include a new region. All loads and stores to this region would be interpreted and handled by the Recryptor backend. This effectively models the memory-mapped decoder that we implemented in the test chip. Because every Cortex M0 and Recryptor operation is of fixed latency (as there are no non-uniform memory access effects), modeling performance with high accuracy is very straightforward.

There are two ways to trigger computation with Recryptor. The first way is to program a task specification, which includes the desired operations, base addresses, and number of banks/words to use, into Recryptor via the memory-mapped decoder. In this way, Recryptor runs operations one at a time, with each subsequent operation requiring a new task specification. This mode is called "Recryptor (w/o FSM)" in the following text. The second way is to program a set of task specifications into Recryptor's internal FSMs, so that after being triggered, the FSM will run multiple operations back-to-back without requiring reprogramming in between. This mode reduces Recryptor launch overhead and provides significant speedup over the first mode. In the following text, this second mode is called "Recryptor (w/FSM)". The L*ó*pez-Dahab finite field multiplication is one example of a workload that benefits greatly from this FSM mode.

The implementation of this simulator is available on GitHub [80].

Table 5.1: Number of cycles for Finite Field Multiplication and Reduction

#Cycles (Speedup)	163-bits	233-bits	283-bits	409-bits
Cortex-M0	7904 (1×)	10152 (1×)	12485 (1×)	19598 (1×)
Recryptor (w/o FSM)	1962 (4.0×)	2492 (4.1×)	2742 (4.6×)	3682 (5.3×)
Recryptor (w/ FSM)	905 (8.7×)	1157 (8.8×)	1284 (9.7×)	1789 (11×)

5.3 Case study: Elliptic Curve Arithmetic

In this section, we evaluated the performance gains of Recryptor on elliptic curve arithmetic operations, specifically binary Koblitz curves of order 2^{163}, 2^{233}, 2^{283}, 2^{409}. We use the RELIC toolkit [5] for the baseline software implementation. To evaluate Recryptor speedups, we make the appropriate modifications to replace specific software functions with Recryptor-accelerated functions.

5.3.1 Field Arithmetic

For field multiplication, the L*ó*pez-Dahab left-to-right comb method is used with a window of ω=4; Reduction is done with the fast reduction method; Inversion is performed with the Extended Euclidean algorithm.

Table 5.1 compares the performance (in cycles) for a finite field multiplication between the baseline Cortex-M0, Recryptor without FSM, and Recryptor with FSM. Without using the FSM, Recryptor provides between 4-5.3× speedup over the baseline Cortex M0 across the range of 163-409 bits. However, by applying the FSM, this speedup increases to between 8.7-11×.

The difference between the simulation data (Table 5.1) and test chip data (Table 4.4) is mostly due to the fact that the test-chip data does not include memory move operations that are needed to satisfy data-alignment requirements of Recryptor's in-memory compute operations, but the simulation data includes this overhead.

Table 5.2: Number of cycles on $\mathbb{F}_{2^{163}}$ Koblitz curve

#Cycles (Speedup)	Point addition	Random point mult	Fix point mult
Cortex-M0	44491 (1×)	3656365 (1×)	3405931 (1×)
Recryptor (w/o FSM)	14584 (3.1×)	1970110 (1.9×)	1852503 (1.8×)
Recryptor (w/ FSM)	9281 (4.8×)	1670827 (2.2×)	1576869 (2.2×)

Table 5.3: Number of cycles on $\mathbb{F}_{2^{233}}$ Koblitz curve

#Cycles (Speedup)	Point addition	Random point mult	Fix point mult
Cortex-M0	55241 (1×)	5921036 (1×)	5607359 (1×)
Recryptor (w/o FSM)	16460 (3.4×)	2851831 (2.1×)	2697891 (2.1×)
Recryptor (w/ FSM)	9950 (5.6×)	2334388 (2.5×)	2206819 (2.5×)

5.3.2 Curve Arithmetic

López-Dahab and affine mixed coordinates are used for point addition. Point multiplication uses TNAF with the sliding window method. For both fixed-point and random-point multiplications, the window parameter is set to ω=4.

Tables 5.2, 5.3, 5.4 and 5.5 compare the performance (in cycles) of point addition as well as random-point and fixed-point multiplications of binary Koblitz curves for each of the three scenarios. For binary elliptic curve with 163 to 409 bits, Recryptor (w/o FSM) has speedup of 3.1-4.2× on point addition, 1.9-2.6× on random point multiplication and 1.8-2.6× on fix point multiplication; while Recryptor (w/ FSM) has speedup of 4.8-6.9× on point addition, 2.2-3.1× on random point multiplication and 2.2-3.2× on fix point multiplication. For 233 bits, the average number of finite field multiplications are 5 for point additions, 397 for random-point multiplications and 377 for fixed-point multiplications. The speedups measured in simulation are in good agreement with test-chip data (Table 5.1).

5.4 Case study: AES-GCM

To run AES-GCM, we need an efficient right-shift for multiplication. However, the Recryptor test-chip only supports a left-shift operation. In the simulator, we augment Recryptor

Table 5.4: Number of cycles on $\mathbb{F}_{2^{283}}$ Koblitz curve

#Cycles (Speedup)	Point addition	Random point mult	Fix point mult
Cortex-M0	69184 (1×)	9072592 (1×)	8686064 (1×)
Recryptor (w/o FSM)	10394 (6.7×)	4557208 (2.0×)	4282838 (2.0×)
Recryptor (w/ FSM)	13002 (5.3×)	3800480 (2.4×)	3645475 (2.4×)

Table 5.5: Number of cycles on $\mathbb{F}_{2^{409}}$ Koblitz curve

#Cycles (Speedup)	Point addition	Random point mult	Fix point mult
Cortex-M0	104142 (1×)	17714966 (1×)	16987883 (1×)
Recryptor (w/o FSM)	24534 (4.2×)	6932240 (2.6×)	6621715 (2.6×)
Recryptor (w/ FSM)	15069 (6.9×)	5650537 (3.1×)	5390265 (3.2×)

with the necessary shifter functionality to efficiently run AES-GCM. This is the only change that is required to run AES-GCM. The results of this workload across a varying number of input bytes is shown in Table 5.6. Our setup has similar results compared with the reference baseline ([13]), when running the CIFRA library, as shown of column 2 and 3 in the table. For inputs with 4 to 64 bytes, Recryptor has speedup of more than 12.8× when compared with the reference baseline.

5.5 Conclusion

Point multiplication is a key step for elliptic curve based cryptographic protocols, but it is very expensive to compute in software, taking millions of cycles for fundamental operations on Cortex M0 platforms. These operations are well suited for hardware acceleration, but evaluating larger workloads in RTL simulation is infeasible. In this chapter, we proposed Recryptor M-ulator, a simulator that can be used to perform design space explorations and evaluate the speedup of the Recryptor architecture on large workloads like AES-GCM and elliptic curve arithmetic.

Table 5.6: Number of cycles for AES- GCM

Input	**CIFRA ([13])**	**CIFRA (Our setup)**	**Recryptor**	**Speed up (Recryptor / [13])**
(Bytes)	#Cycles	#Cycles	#Cycles	-
4	57345	62278	4465	12.8
8	57137	63431	4465	12.8
12	56863	64067	4465	12.7
16	56551	64960	4444	12.7
20	77169	85554	5552	13.9
24	76532	85301	5552	13.8
28	76258	86232	5552	13.7
32	75286	85906	5571	13.5
36	95772	106018	6664	14.4
40	95333	106209	6664	14.3
44	94432	106030	6664	14.2
48	93097	105852	6663	14.0
52	113319	126038	7738	14.6
56	113342	126895	7738	14.6
60	112045	126938	7738	14.5
64	124186	126834	7741	16.0

CHAPTER VI

iRazor: A Current-Based Error Detection and Correction Scheme for PVT variation

This chapter presents iRazor, a lightweight error detection and correction approach to suppress the cycle time margin that is traditionally added to VLSI systems to tolerate process, voltage and temperature variations. iRazor is based on a novel current-based detector, which is embedded in flip-flops on potentially critical paths. The proposed iRazor flip-flop requires only three additional transistors, yielding only 4.3% area penalty over a standard D flip-flop.

The proposed scheme is implemented in an ARM Cortex-R4 microprocessor in 40nm through an automated iRazor flip-flop insertion flow. To gain an insight into the effectiveness of the proposed scheme, iRazor is compared to other popular techniques that mitigate the impact of variations, through the analysis of the worst-case margin in 40 silicon dies. To the best of the authors' knowledge, this is the first work that compares the measured cycle time margin and the power efficiency improvements offered by frequency binning and various canary approaches. Results show that iRazor achieves 26%-34% performance gain and 33%-41% energy reduction compared to a baseline design across the 0.6 to 1V voltage range, at the cost of 13.6% area overhead.

6.1 Motivation

Processors and Systems-on-Chip (SoCs) are traditionally designed to accommodate for worst-case variations, with a cycle time target that incorporates process, voltage, temperature, and aging (PVTA) margins, which in turn substantially degrade performance and energy efficiency. Adaptive designs with *in-situ* error detection and correction capability have been widely explored to suppress the cycle time margin, using specialized registers on critical paths that perform timing error detection and correction (EDAC) [15–19, 23, 40, 52, 57]. Unfortunately, such specialized registers typically incur a large area overhead compared to conventional registers. For example, Razor requires 44 extra transistors per register [23], double sampling with time borrowing (DSTB) [15] needs 26 extra transistors, and Razor-lite [40] requires 8 extra transistors, which is currently the EDAC approach with smallest overhead. The significant area overhead has been an obstacle to the adoption of EDAC approaches in commercial designs, and currently there is no significant commercial processor implementing EDAC approaches [4]. In addition, the performance and energy gains from EDAC approaches have not been thoroughly quantified in relation to competing approaches to mitigate variations at lower overhead, such as frequency binning, critical path monitors [14, 22, 34, 66] and canary circuits [71].

6.2 Previous Works

Traditionally, processors are margined to tolerate process, voltage and temperature (PVT) variations. Among the existing techniques to reduce their impact on the related cycle time margin, frequency binning entails the lowest overhead as it relies on additional testing time to perform coarse-grained discrete frequency tuning to mitigate process variations at given environmental conditions.

More sophisticated self-adapting design techniques introduce process and environmental sensors (e.g., ring oscillators) to further reduce the margin, and are customarily adopted in

today's processor and SoC designs. These approaches can adapt to variations to some extent, monitoring them through "canary" circuits that mimic the delay of the critical path(s), and fitting the actual margins. However, the design margin cannot be completely eliminated by these approaches, due to the residual mismatch between the on-chip sensors (e.g., ring oscillator frequency) and the actual critical path delay.

EDAC approaches can virtually eliminate the design margin, based on the insertion of specialized registers on critical paths to perform timing error detection and correction. Among the proposed techniques, output waveform analysis [26], time-redundant latches [54], transition detector with time borrowing (TDTB) [15], DSTB [15] and different Razor latches [17, 19, 25, 40] have been proposed. For example, the Razor approach eliminates the design margin by allowing for reducing the clock cycle until timing constraints are barely met. This occurs right before timing failures are detected by specialized registers, such as Razor-I [23], Razor II [19], Bubble-Razor [25] and Razor-lite latches [40]. The key idea of Razor latches is that the data comes into the main flip-flop and is also tapped off to a shadow latch, which is clocked slightly later. The mismatch between the output of the main flip-flop and the shadow latch reveals the occurrence of the timing error. Once an error is detected, it can be corrected in several manners as proposed in previous work. For example, [25] uses a bubble propagation algorithm to send stalling signals to neighbors in half a cycle assuming a 2-phase latch clocking. As another example, global clock gating and counterflow pipelining were proposed in Razor I [23]. In the former technique, the whole processor is stalled until correct values are reloaded. Through counter-flow pipelining, a bubble is sent upstream and downstream pipeline stages at every clock cycle to prevent the propagation of errors and perform their correction.

Although EDAC techniques fundamentally eliminate the design margin, they suffer from relatively large area and energy overhead due to the complexity of the detection mechanism. For example, [15–17, 19, 23, 40] require eight additional transistors per flip-flop or more. The direct and significant impact on cost has limited the diffusion of prior EDAC techniques,

as confirmed by the lack of adoption in any significant commercial design to date, and motivates the introduction of novel lightweight EDAC schemes that can be truly afforded in real designs.

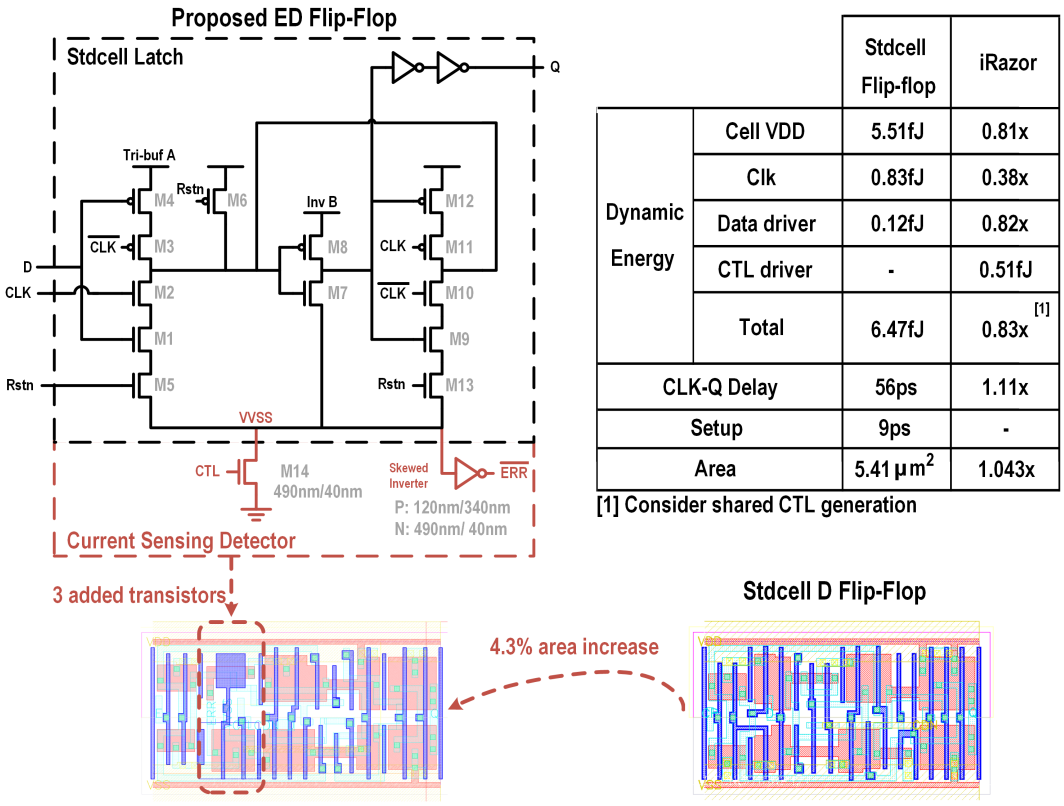
6.3 Proposed iRazor Circuit and Analysis

6.3.1 The iRazor Flip-flop

The iRazor Flip-Flop supplements a latch circuitry [70] with asynchronous reset (signal Rstn) in Figure 6.1 (drawn in black) with the lightweight error detection circuit (highlighted in red). The latter consists of a novel 3-transistor current detector that reveals whether the latch is transiently drawing any transistor on-current after the clock edge, thus effectively detecting transitions occurring at the input of the iRazor flop. In the following, positive edge-triggered timing is assumed with no loss of generality.

Timing violations are caught within an error detection window during which the first tristate inverter (M1-M5 in Figure 6.1) is transparent, and it represents the portion of the clock cycle when the input should not transition to avoid timing violations. The detection window is defined by setting the signal CTL in Figure 6.1 as low, and timing violations are signaled by the active-low error signal $\overline{ERR}$ in Figure 6.1. As discussed below, the error detection window starts after the falling edge of CTL, thus enabling some amount of time borrowing at the very beginning of the clock cycle, in addition to the capability of subsequently detecting timing violations.

When the iRazor input correctly transitions before the rising clock edge and after the falling clock edge as in Figure 6.2a, CTL is high and transistor M14 in Figure 6.1 is ON, thus tying the virtual ground (VVSS) to ground. Accordingly, the iRazor circuit in 6.1 operates like a conventional flip-flop and updates its output at the rising clock transition, which makes the first tristate inverter transparent. In this case, the active-low error signal $\overline{ERR}$ is deasserted (i.e., $\overline{ERR}$ is set to 1) by the skewed inverter in red, as required. Instead,

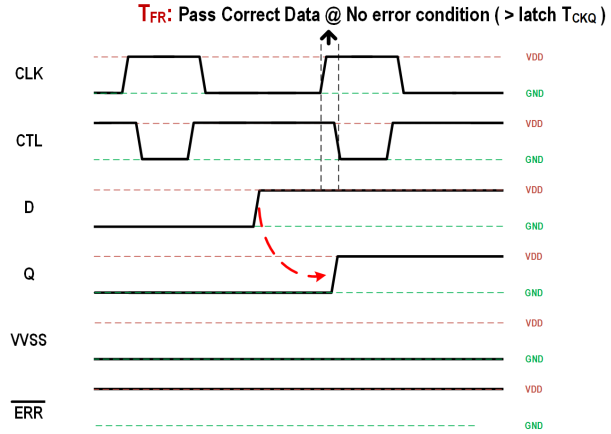


		Stdcell Flip-flop	iRazor
Dynamic Energy	Cell VDD	5.51fJ	0.81x
	Clk	0.83fJ	0.38x
	Data driver	0.12fJ	0.82x
	CTL driver	-	0.51fJ
	Total	6.47fJ	0.83x [1]
CLK-Q Delay		56ps	1.11x
Setup		9ps	-
Area		5.41 μm^2	1.043x

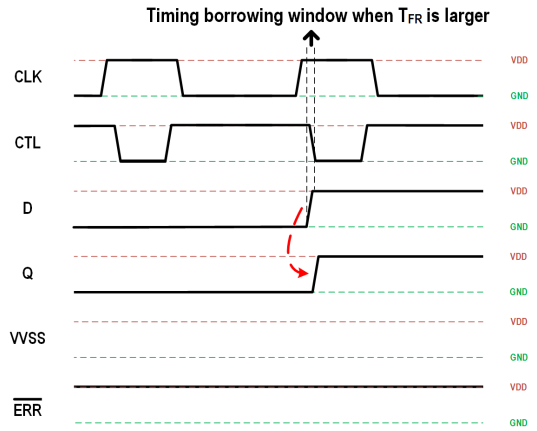
[1] Consider shared CTL generation

Figure 6.1: Schematic of the proposed iRazor flip-flop with error detection capability, and its energy, delay and area compared with conventional flip-flop standard cell (both positive edge triggered)

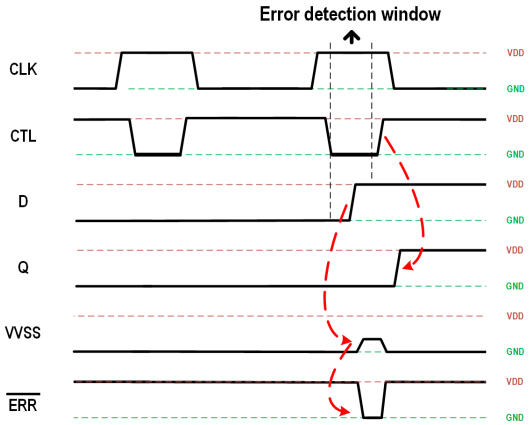
when the iRazor input D transitions after the rising clock edge and before the beginning of the error detection window as in Figure 6.2b, the iRazor latch is transparent and allows for timing borrowing. In this case, moderately late arriving inputs are forgiven and no error is flagged (i.e., $\overline{ERR}$=1).



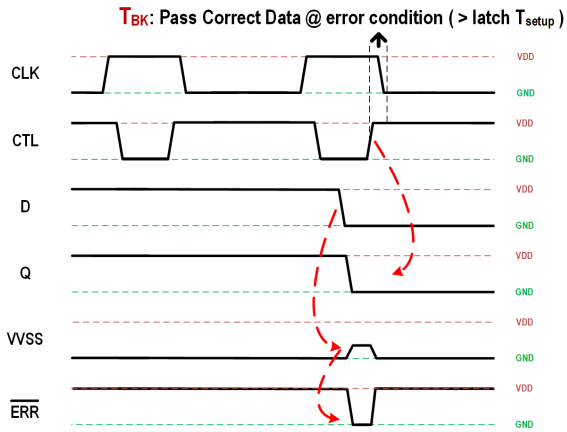
(a) Input D is correctly switching before the rising clock edge



(b) D is switching within the time borrowing window



(c) An error is occurring due to the transition of the input D from 0 to 1 during the error detection window



(d) An error is occurring due to the transition of the input D from 1 to 0 during the error detection window

Figure 6.2: Waveforms in iRazor flip-flop

During the error detection window as in Figure 6.2c, 6.2d, the *CTL* signal is set to 0, transistor M1 is turned off and the virtual ground is disconnected from the ground. If no input transition occurs during the error detection window, the virtual ground is dynamically

held at ground, and no error is flagged by the skewed inverter in 6.1 (i.e., $\overline{ERR}$ is kept at 1). Instead, if the flip-flop input D performs a transition during the error detection window, the voltage of the floating virtual ground is raised by the charge provided by either the first tristate inverter (M1-M5) or the subsequent inverter (M7-M8), as discussed in the following. The red inverter in 6.1 is skewed low so that the raised virtual ground voltage lies beyond the inverter logic threshold, and hence $\overline{ERR}$ is set to 0, thus signaling an error. In particular, if D transitions from 0 to 1 during the error detection window (see 6.2c), the initially discharged capacitance at the virtual ground node VVSS is charged by transistors M1-M2 and M5. This is due to the charge sharing with the capacitance at the output of the tristate inverter M1-M5, which was precharged at V_{DD} by M1-M5 before the input transition, since the input D was initially equal to 0. Similarly, when D transitions from 1 to 0 during the error detection window (see Figure 6.2d), the capacitance at the virtual ground node is charged by transistor M7, due to the charge sharing with the capacitance at its output. In both cases, the virtual ground voltage VVSS is raised and complemented by the skewed inverter in Figure 6.1 to flag the error and hence set $\overline{ERR}$ to 0. According to the above considerations, the VVSS node is dynamic and its signal integrity needs to be preserved through routine layout strategies, such as shielding or proper spacing of strong aggressors.

To ensure correct error detection, the error detection window has to be correctly aligned with the clock cycle. In particular, from Figure 6.2a the falling edge of CTL marks the start of the detection window and must occur with sufficient delay after the rising clock edge. Otherwise, correct output transitions right after the clock edge would be incorrectly flagged as errors, due to the subsequent transition in the first tristate inverter (M1-M5) occurring a clock-to-Q delay after the clock edge. This minimum delay from the rising clock edge and between the beginning of the error detection window is here referred to as the front timing constraint T_{FR}, and must certainly exceed the flip-flop clock-to-Q delay to allow the data to pass through the slave latch M9-M13 without triggering an error. Larger values of T_{FR} allow time borrowing as in Figure 6.2b, although at the expense of a shorter error detection

window.

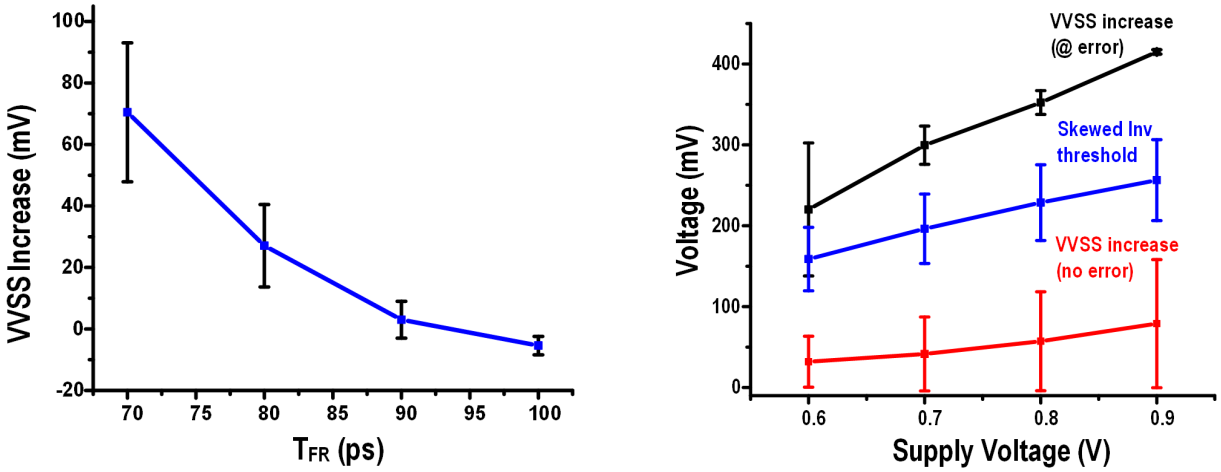
To assure that the input data is correctly latched into the cross-coupled inverter pair (M7-M13 in Figure 6.1) during the error detection window, the latter needs to end before the falling clock edge by an appropriate back time constraint T_{BK} as in Figure 6.2c, 6.2d. Quantitatively, T_{BK} needs to be greater than (or equal to) the latch setup time T_{setup}, so that metastability is prevented during the error detection window.

6.3.2 Analysis of Robustness, Area and Energy

In general, increasing T_{FR} leads to a wider time borrowing window at the expense of a shorter error detection window. Also, larger T_{FR} reduces the probability of false positive errors due to the transition in the output of the first tristate inverter M1-M5 right after the rising clock edge, and ending some time after a clock-to-Q delay (i.e., when the output of the tristate inverter is close to the steady state). More quantitatively, T_{FR} needs to be large enough to give transistor M14 enough time to bring the virtual ground VVSS back to the ground voltage (since CTL=1), after its temporary increase due to the above transition in M1-M5.

Monte Carlo simulations in Figure 6.3a illustrate the relationship between T_{FR} and the VVSS increase during time borrowing (i.e., when no error occurs), including variations. From this figure, large enough T_{FR} values keep the VVSS upward transition small when no error occurs. As shown in Figure 6.3b, large enough values of T_{FR} make the temporary VVSS increase caused by data transitions in the time borrowing window smaller than the skewed inverter threshold voltage, and avoid false error triggering (shown by the blue line). In case of timing error occurrence (see black line in Figure 6.3b), the VVSS increase exceeds the skewed inverter threshold voltage to trigger an error. However, the ability to detect an error is potentially compromised at very low voltages, for a given T_{FR}. For example, Figure 6.3b shows that some error may not be occasionally flagged at 0.6V and below, as the VVSS increase might be higher than the skewed inverter logic threshold in some

rare cases. Indeed, the whiskers of VVSS and the threshold of the skewed inverter start overlapping at 0.6V in Figure 6.3b.



(a) No Error Operation

(b) Error Detection

Figure 6.3: Statistical analysis of a) virtual ground voltage under errors vs. T_{FR} (1,000 Monte Carlo runs). Whiskers indicate 3 standard deviations around the mean value

Results of post-layout analysis of the iRazor flip-flop relative to a standard flip-flop are reported in Figure 6.1. The added 3 transistors in red in Figure 6.1 increase the area by 4.3%, due to the large gate length of the PMOS transistor in the skewed inverter, as required to make its logic threshold closer to ground to better capture the VVSS increase. In the adopted technology, increasing the gate length of PMOS to reduce the logic threshold is preferable to stacking, as the latter would entail a larger area penalty of 11.8%. The total dynamic energy of the iRazor flip-flop is decreased by 17% compared to the conventional flip-flop, when sharing the *CTL* generation circuitry, as discussed in the final chip implementation in Section 6.4. This figure also gives the breakdown of the energy across cell VDD, clock, input driver and *CTL* driver. The iRazor clock-to-Q delay increases by 11% compared with the conventional flip-flop.

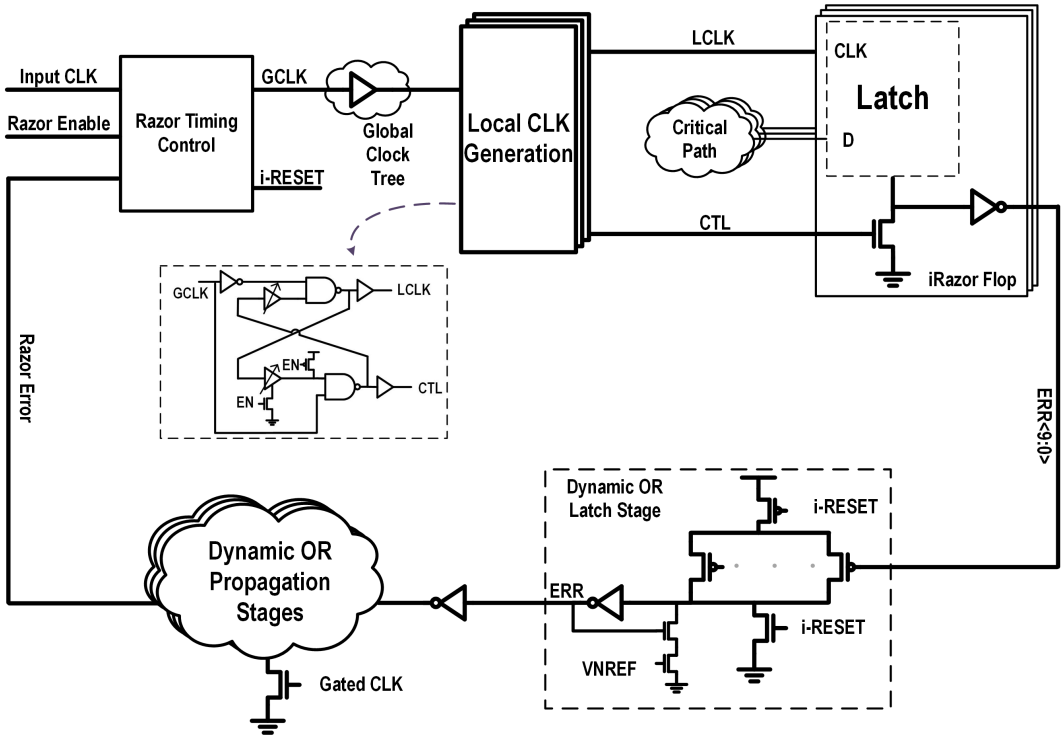


Figure 6.4: Overall iRazor EDAC scheme diagram

6.4 Error Detection and Correction Scheme

This subsection describes the global error detection and correction scheme for iRazor, as shown in Figure 6.4. This is similar to the global clock gating scheme mentioned in Section 6.2. Local clock generators are used as the last level of the clock tree to generate the clock and the *CTL* signals in iRazor, as shown in Figure 6.4. These generators are shared between registers to minimize the area and energy overhead, and control the T_{FR} and T_{BK} windows in Section 6.3 to avoid the power overhead and the inter-clock skew that would be needed by two clock distribution networks.

Under normal operation when no error occurs, data arrives before the rising clock edge and the iRazor output Q latches the value after the clock rises, with $\overline{ERR}$ staying high. When an error occurs due to a data transition within the detection window, the $\overline{ERR}$ signal is pulled low by the skewed inverter in Figure 6.1 of the relevant flip-flop. The resulting $\overline{ERR}$ signal experiences a negative pulse, which is captured by a PMOS-based dynamic OR-latch,

which is shared by up to 10 iRazor flip-flops, as shown in Figure 6.4. The aggregate output of the OR-latch is then ORed together with all other aggregate error signals within the processor by using conventional dynamic OR gates, thus generating the global Razor error signal in Figure 6.4. This global Razor error signal then propagates through the Razor timing control in the same figure. Razor timing control skips the clock edge following the occurrence of an error, providing the pipeline with a further cycle to resolve the error, as shown in the third cycle at the left of Figure 6.5 (the error occurs in the second cycle). Following error resolution, the dynamic OR-latches are reset using the i-RESET signal. . Normal operation resumes in the next cycle (fourth cycle in Figure 6.5), as the global razor error signal is reset to 0 when clock gating is released. The dynamic OR latch stages (Figure 6.4) are reset through the i-RESET signal, which can catch the $\overline{ERR}$ signal generated by the iRazor flip-flop when the clock is either low or high. The dynamic OR propagation stages are reset using the gated CLK signal to keep the global Razor error signal to be high within the error recovery stage (in Figure 6.5) to avoid glitches of the gated local clocks.

Using local detection and clock stalling, the pipeline is halted within one cycle of a detected error, allowing the EDAC technique to be integrated into the processor without requiring rollback or architectural changes. To accomplish this, the error signal must propagate through the above logic within one clock cycle. As shown at the right of Figure 6.5), the error critical path includes: the clock tree delay to reach the clock tree leaves from the clock root first, then the T_{FR} delay, the detection window itself, the error detection delay, the dynamic OR latch stage and three dynamic OR propagation stages, and finally the Razor timing control to ultimately generate the clock gating signal.

6.5 Automated iRazor Design Flow

The automated and architecture-independent iRazor flow in Figure 6.6 was developed and adopted to design an ARM Cortex-R4 processor, which is used as reference design example in the following.

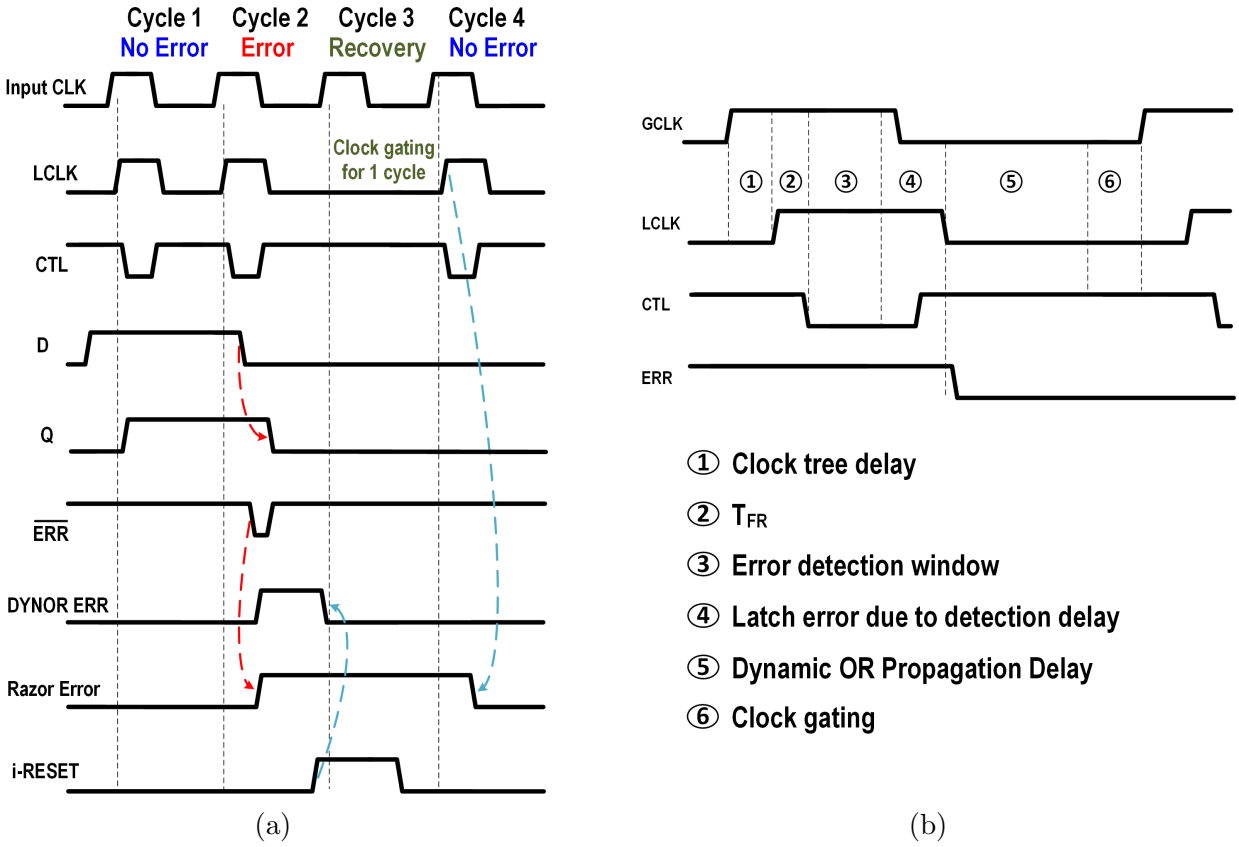


Figure 6.5: (a)iRazor timing diagram; (b) timing analysis of the error critical path

The iRazor design flow starts with a placed and routed baseline design. Then, flip-flops to be razorized are selected, based on the tradeoff between the path coverage and the area overhead due to iRazor flip-flops, the transistor upsizing to meet timing, and the additional hold buffers, which are required to make the min-delay larger than the transparency window in the covered paths. As shown in Figure 6.7, iRazor flip-flops are progressively inserted to cover paths with increasing timing slack (i.e., from the most to the least critical one), and higher path coverage entails a larger number of iRazor flip-flops and area. A high path coverage also makes the design hard to route. In the considered ARM Cortex-R4 design, from Figure 6.7 a reasonable compromise between path coverage and overhead is to cover paths with 200 ps timing slack or lower, replacing the corresponding conventional flip-flops with iRazor flip-flops. This leads to the replacement of 8.7% of the total flip-flop count. The resulting datapath delay histogram after razorizing is shown in Figure 6.8 together with the

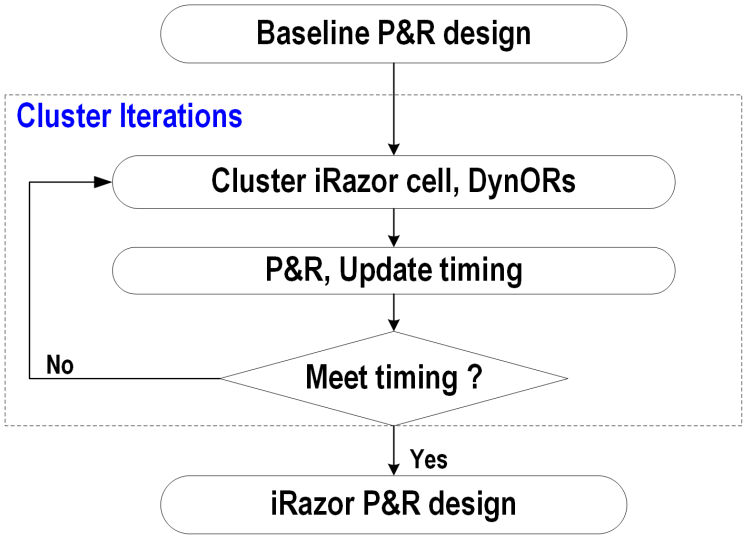


Figure 6.6: Architecture-independent automated flow for iRazor flip-flop replacement and clustering

baseline histogram. Overall, path delays in iRazor are pushed to the right because of the addition of hold buffers. The last two columns represent paths with iRazor flops.

After iRazor insertion, placement of dynamic ORs needs to be optimized. According to the initial placement of the baseline design, automated clustering of iRazor cells is performed to share the local clock generator and the different levels of dynamic OR trees. Both the physical locations and the loading in each stage are key factors for clustering. A threshold distance is set first for the iRazor flip-flop clustering into the same group, creating a new group once the threshold is exceeded. In this design, the distance threshold is set to 60 μm, 300 μm and 1,000 μm for the first, the second and the third stage. Figure 6.9 shows the resulting placement of iRazor flip-flops, dynamic OR latches and subsequent stages.

Then, place and route is performed, checking the timing of the overall error control feedback loop since the wirelength from the skewed inverter output to the dynamic OR-latch is critical for timing closure (see Figure 6.5 on the right). If timing is not met, hierarchical iterations of clustering are performed followed by a new placement, while freezing the original iRazor flip-flop locations to facilitate convergence. Further iterations of clustering/placement are performed until the timing is closed. Then, a final iRazor place and routed design is

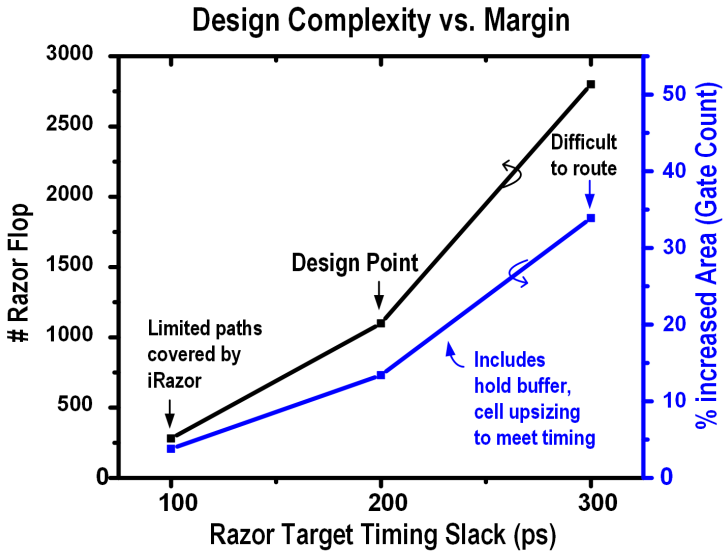


Figure 6.7: Design complexity (in number of iRazor flip-flops) vs. targeted timing slack of iRazor

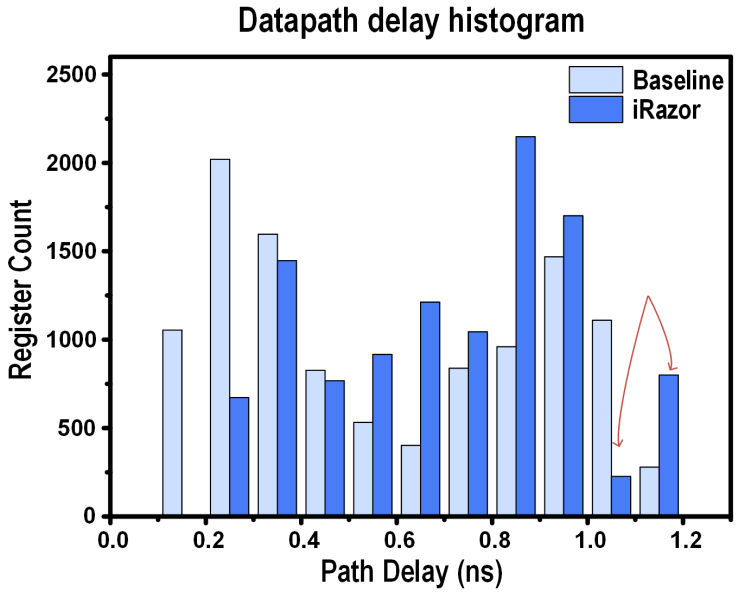


Figure 6.8: Path delay histogram of baseline and iRazor design

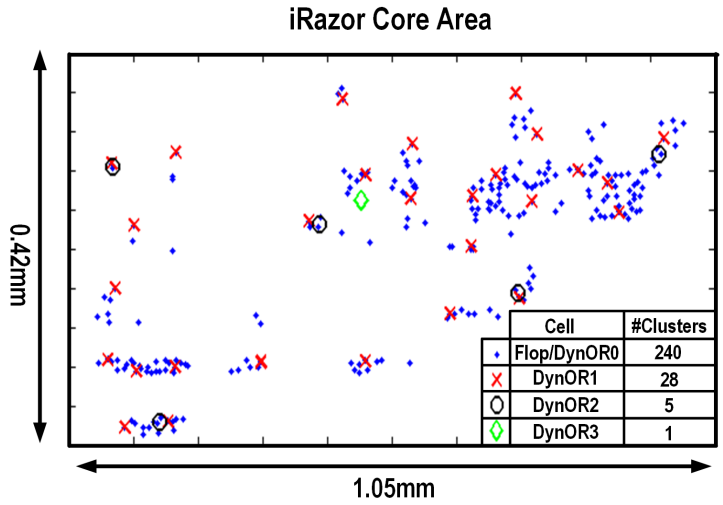


Figure 6.9: iRazor cluster spatial position within the on-die processor footprint

	3	Additional transistors over latch
-	8	Latch compared with flip-flop
+	30*240/1115	240 blocks of local clock generation over 1115 iRazor flops
=	1.46	

Figure 6.10: iRazor effective overhead explicit calculation

achieved, with all prior steps performed in a fully automated fashion. As well known for all EDAC approaches, timing closure might not be guaranteed in very large designs, although iRazor is demonstrated to work in a microprocessor core that is an order of magnitude more complex than prior demonstrations (see Figure 6.1).

The effective overhead of the iRazor scheme relative to a conventional flip-flop based design is shown in Figure 6.10. First, three additional transistors are included in each latch, although the latch itself has eight fewer transistors than a conventional flip-flop. Then, 240 local clock generation blocks are used in the final design, each comprising 30 transistors. The additional transistors are amortized across the 1,115 iRazor flip-flops, resulting to an effective overhead of only 1.46 transistors per flip-flop.

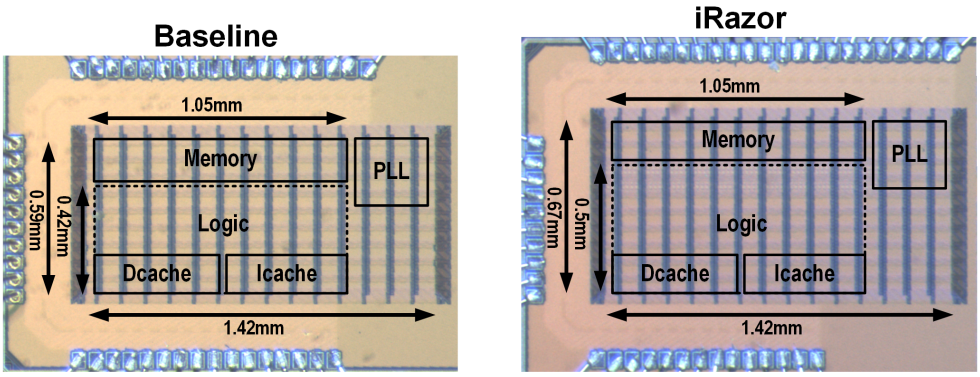


Figure 6.11: Die photo of baseline and iRazor Cortex-R4 processor in 40nm CMOS

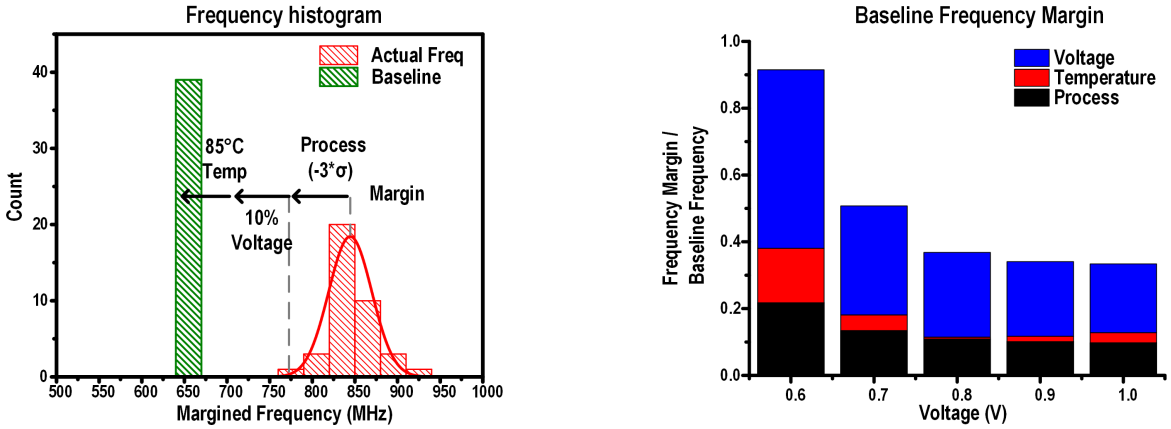
6.6 Testchip

Both the baseline and the iRazor designs of the targeted processor were implemented on a testchip, whose micrograph is shown in Figure 6.11. The ARM Cortex-R4 processor was implemented in 40nm CMOS, with a total number of flip-flops of approximately 13,000, of which 8.7% were razorized. The total number of gates increased by 13.4% when applying iRazor, due to the addition of minimum-sized hold time buffers, iRazor flip-flops, the OR tree, and the *CTL* tree, which respectively contributed by 10.06%, 0.95%, 0.27 and 0.36% to the overall area increase, while the remaining 1.76% is due to signal routing. The total iRazor core area includes 8-KB instruction/data cache and 12-KB memory, and increased by about 13.6% compared to the baseline. Note that buffer insertion takes most of the area in logic in this specific design, although the memory size can be much larger in many other modern processors, in which case the percentage overhead is expected to be significantly reduced. Compared with previous EDAC testchips, this design marks a significantly more complex processor implementation, particularly in terms of the number of total and replaced flip-flops, other than gate count.

6.7 Comparisons with Binning and Canary Techniques

Based upon the techniques discussed in Section 6.2, 40 baseline chips were measured to gain an insight into the effectiveness of iRazor, compared to a baseline margined design, frequency binning and ring oscillator-based canary methods.

The worst-case margining of 85°C temperature, 10% supply drop, and 3σ process variation is used to define the baseline. As shown in Figure 6.12a, the histogram in red is the maximum operating frequency of 40 baseline chips at 1V and room temperature, whereas the margined frequency able to work across all PVT variations is plotted in green. The margined frequency is typically 25% lower and up to 32% than the maximum frequency allowed by the measured chips. The detailed margin breakdown into process, voltage and temperature across 0.6-1V is plotted in Figure 6.12b, which shows that voltage margin gives the largest contribution. As the processor voltage approaches the threshold voltage, the margin contributions increase substantially (i.e., 2X or more).



(a) (b)

Figure 6.12: (a) Detailed frequency histogram and margin analysis of baseline at 1V; (b) Baseline frequency margin across 0.6 1V voltage range, including 10% voltage margin, 60°C temperature margin, 3 sigma process margin. The frequency margin is normalized to the average across dice of its actual frequency at nominal voltage/temperature conditions.

Let us now consider the case of frequency binning, with dies being divided into three bins based on their process corner labeled as slow, typical and fast in Figure 6.13. Then,

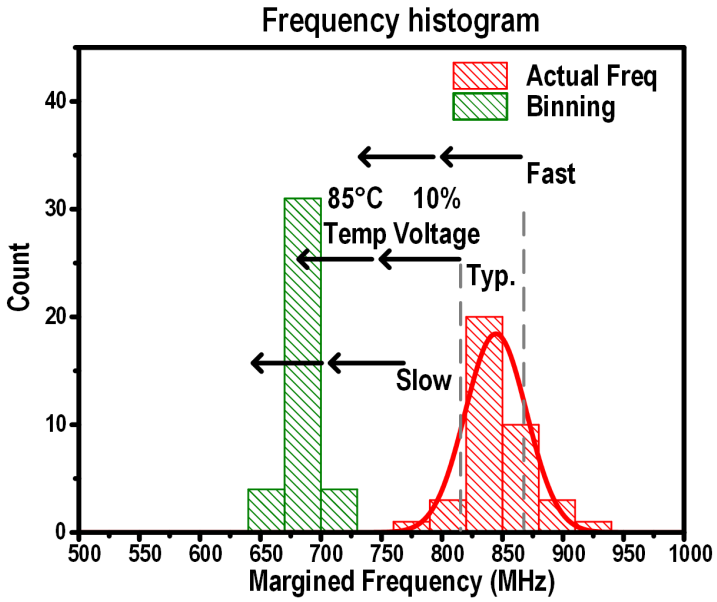
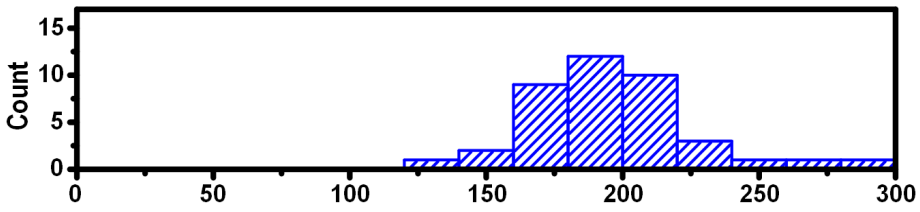


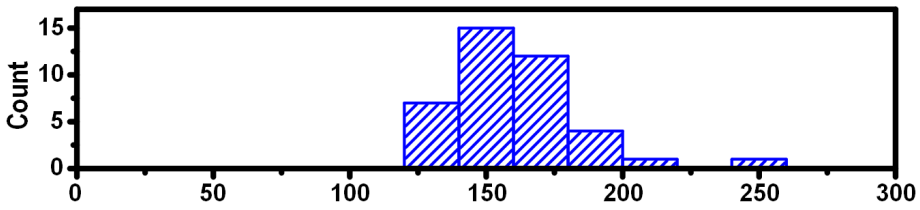
Figure 6.13: Detailed frequency histogram and margin analysis of frequency binning method at 1V

each bin is margined for worst-case temperature and voltage (85°C, 10% supply drop). The frequency histogram under frequency binning for the 40 chips at 1V is shown in Figure 6.13, whose comparison with Fig. 12a clearly shows that some margin is removed from the baseline approach. For completeness, the detailed margin histogram of the frequency binning is shown in Figure 6.14b.

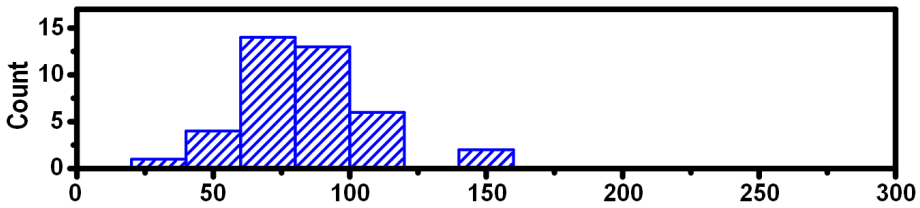
As third variation-aware mainstream design approach, let us consider the "simple canary" method, under which the baseline processor is equipped with a ring oscillator (RO) used as processor frequency predictor. Figure 6.15 shows measured processor frequency versus RO frequency across 0.6-1V and 20-85°C. Exploiting the correlation between the processor frequency and the RO across voltages and temperatures in the available 40 dice in Figure 6.15, the processor frequency is obtained by fitting the RO frequency data points. 2σ fitting error calculated across dies and PVT conditions is applied to evaluate the RO-processor mis-tracking. In addition, the fitting is de-rated by a 5% voltage margin to account for fast transient voltage excursions that the canary cannot capture. The final frequency histogram of simple canary after including fitting error and the 5% voltage margin is shown in Figure 6.16.



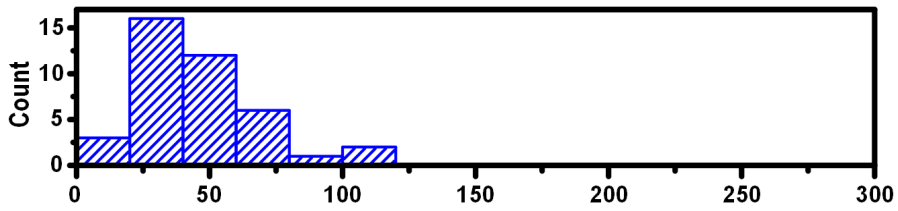
(a) Baseline(process, 10% voltage, temperature)



(b) Binning(10% voltage, temperature)



(c) Simple Canary(process, 5% voltage, temperature)



(d) Canary T/V Spec (process, 10% voltage)

Figure 6.14: Margin histogram for different methods (1V, room temperature).

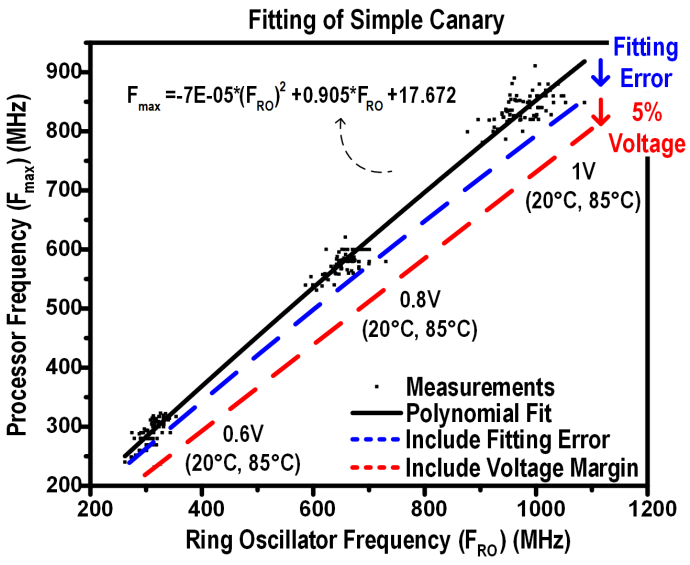


Figure 6.15: Fitting of operating frequency vs. ring oscillator frequency in simple canary fitting method

The margin histogram of the simple canary approach is also shown in Figure 6.14c.

A further comparison, a less simplistic canary approach is considered where each data point is treated as a temperature/voltage-specific canary, to suppress the margin due to temperature and voltage. This is customarily achieved by introducing on-die temperature and voltage sensors, which quantify temperature and voltage of each data point. In this approach, the linear correlation between processor and RO frequency is determined for each temperature and voltage condition. The measurements of 0.6, 0.8 and 1V and the fitting to the ring oscillator frequency are shown in Figure 6.17, where blue dots refer to 25°C and the red ones refer to 85°C. The linear fit is again de-rated with 5% voltage margin and 2% fitting error, but here the latter is computed only across dies (i.e., without considering voltage and temperature margins). The resulting margin histogram of temperature/voltage-specific canary is shown in Figure 6.14d, which clearly shows a further margin reduction compared to the above variation-aware approaches.

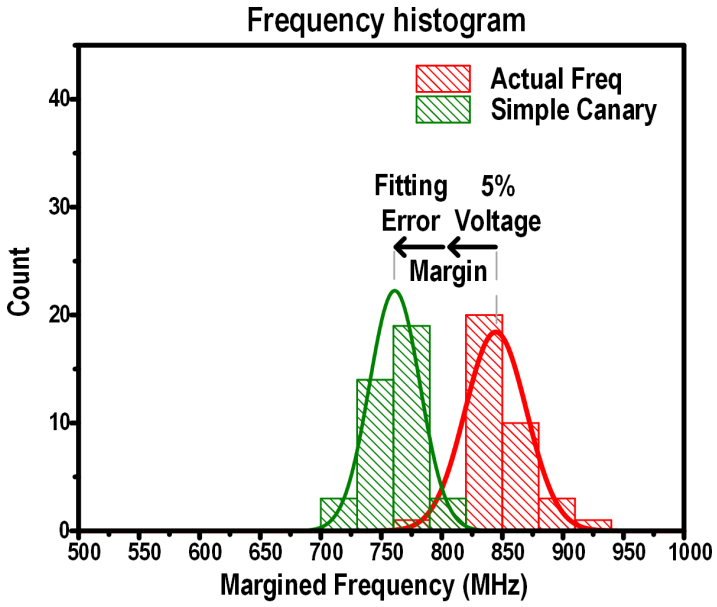


Figure 6.16: Detailed frequency histogram and margin analysis of simple canary method at 1V.

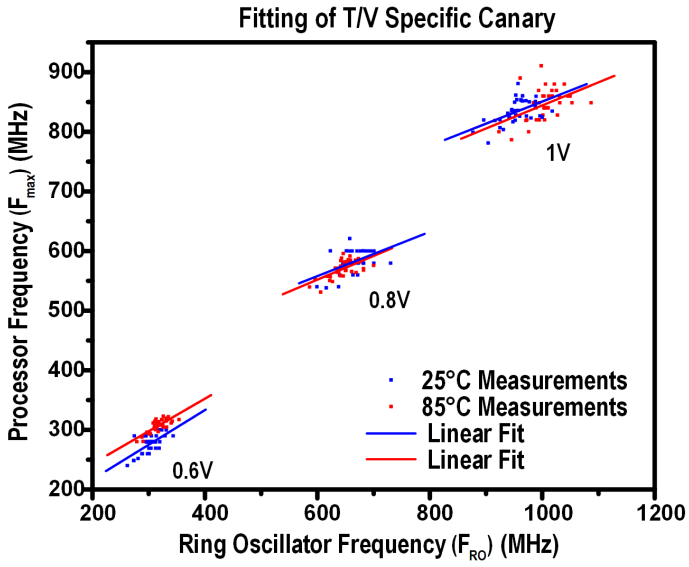


Figure 6.17: Fitting of processor frequency vs. ring oscillator frequency for T/V-specific canary at 25°C and 85°C.

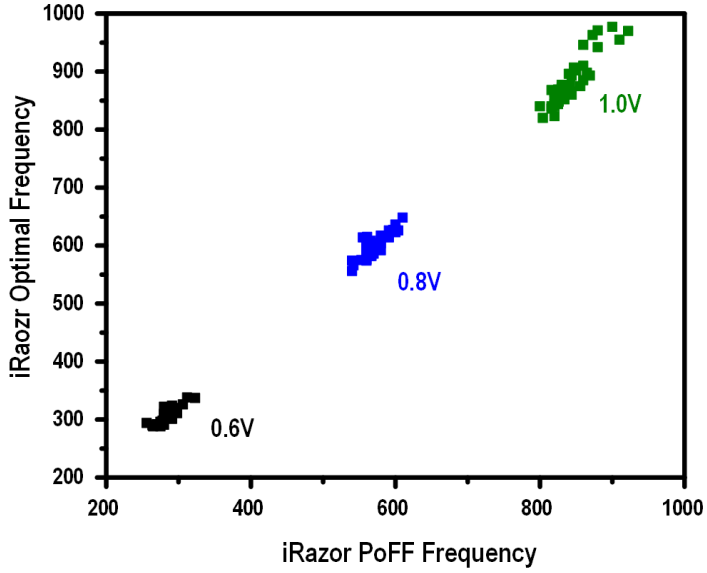


Figure 6.18: iRazor frequency at point of first failure (PoFF) vs. optimal frequency across voltages.

6.8 Experimental Results and Overall Comparison

40 dies of the iRazor design of the ARM Cortex-R4 processor were characterized and compared to the above mainstream variation-aware design methods. The Razor point-of-first-failure (PoFF) frequency is the operating frequency beyond which errors occur (see, e.g., [19] for the details on its measurement). Since iRazor is able to correct errors lying in the transparency window, it can work in a performance-optimal mode where the frequency is pushed beyond the PoFF to allow errors, which are then corrected through the stalling mechanism in Section 6.4. In the performance-optimal mode, the resulting performance includes the effect of both the overscaled frequency and the corresponding stalling cycles due to the resulting errors. The results of the iRazor PoFF frequency and the performance-optimal frequency across 0.6, 0.8 and 1.0V is shown in Figure 6.18. The PoFF represents a conservative 4.4-6.9% timing margin, compared to the performance-optimal iRazor frequency, which corresponds to a 2.4-3% voltage margin. As a comparison, the simple canary approach adds 5% voltage margin to iRazor performance-optimal operating voltage.

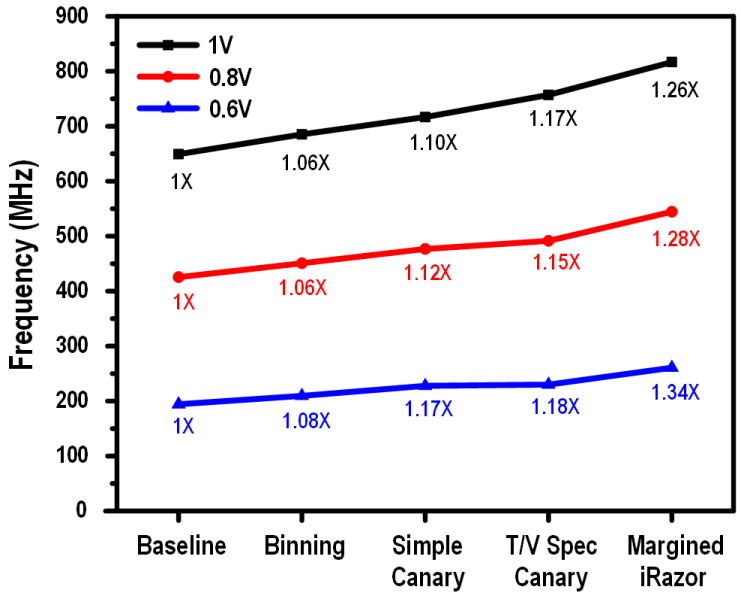


Figure 6.19: Performance comparison between the margined iRazor and other methods across 0.6-1V voltage range.

The previous Razor papers assume that the detection window will surely cover all the PVT variation margins, which is however not always the case. Indeed, the transparency window size depends on the hold margin achieved at design time through the inserted hold buffers, hence practical constraints on the overhead due to the inserted hold buffers may prevent the designer from achieving a detection window that fully covers PVT variations. Therefore, this paper enhances the comparison by considering the margined iRazor frequency, rather than the iRazor PoFF frequency. The maximum frequency allowed by the margined iRazor and all the methods discussed in Section 6.7 is summarized in Figure 6.19. As shown in this figure, a simple canary approach is about twice as effective as binning. The T/V specific canary offers ∼15-18% performance increase over the margined baseline across 0.6V - 1V, while the margined iRazor shows 26-34% performance increase, when considering the same voltage margin as canary methods. This translates into a performance gains of 26%, 19%, and 15% compared to standard, binned, and canary-equipped versions of the Cortex-R4 processor, respectively.

The power consumption at a fixed frequency is compared in Figure 6.20. In this com-

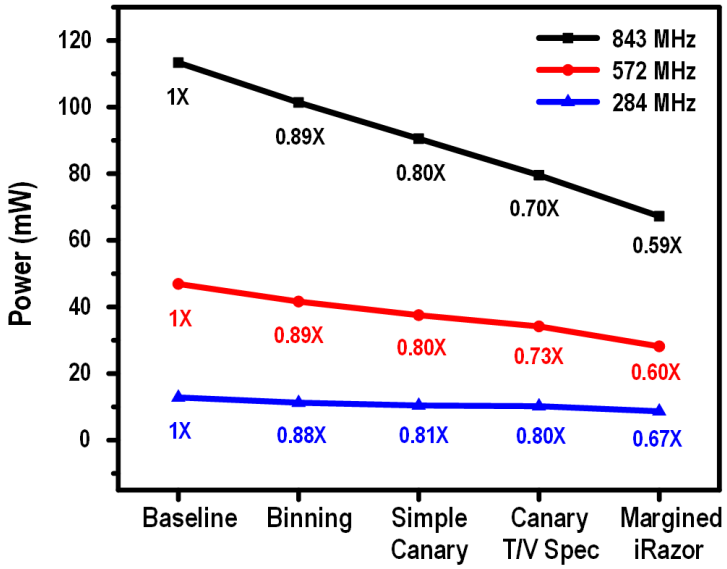


Figure 6.20: Power comparison with the margined iRazor across 0.6 1V voltage range.

parison, we first select the margined baseline frequency at 0.6, 0.8 and 1V as the target, and then we find the required supply voltage to meet this frequency using other techniques. The resulting power for each case is shown in this plot. Simple canary provides a power benefit of ∼20% over baseline across voltage, and the margined iRazor improves power by another 17%∼26% over simple canary from 0.6 to 1V.

As reported in Figure 6.1, iRazor is able to improve the performance by 34% at nominal voltage, and the energy by up to 41% when running at the same performance as the baseline design, thanks to the voltage scaling that it enables.

6.9 Conclusion

The iRazor technique has been proposed as very lightweight technique to enable error detection and correction, with only three additional transistors per flip-flop. An automated design flow assuring time closure has been introduced and applied to implement an ARM Cortex-R4 microprocessor in 40nm. The resulting number of additional transistors compared to a baseline design is 1.54 transistors per flip-flop, which is the lowest reported to date. iRazor has been compared to industry-standard techniques to address variations. iRazor

Table 6.1: Comparison table of EDAC approaches and iRazor.

		Razor II JSSC'09	TDTB JSSC'09	DSTB JSSC'11	ARM JSSC'11	Razor-lite ISSCC'13	iRazor
	Type	Latch	Latch	Latch	Flip-Flop	Flip-Flop	Latch
EDAC Level	**Extra # of Transistor** (I)	31 (8 Shared)	15	26	28+ delay chain	8	1.46 (II)
	Possible Datapath Metastability	No	No	No	Yes	Yes	No
	FF Power Overhead	28.5%	-9%~ -13% (III)	14%~ 34%	Not Reported	2.7%	12.5%
	FF Area Overhead	Not Reported	Not Reported	Not Reported	Not Reported	33%	**4.3%**
Test Chip	**Total Area Overhead**	Not Reported	Not Reported	3.8%	6.9%	4.42%	13.6%
	# Razor cell/ # Total FF	121/ 826	Not Reported	12%	503/ 2976	492/ 2482	1115/ 12875
	# Gate Count	65K (IV)	123K (IV)	Not Reported	Not Reported	Not Reported	1040K
	Technology	130 nm	65 nm	45 nm	32 nm	45 nm	40 nm
	Max. Perf. Improvement (V)	Not Reported	40% @ 0.7V	28% @ 0.8V	50% @ 0.96V	54% @ 0.86V	34% @ 0.6V
	Max. Energy Improvement (VI)	35% @ 185MHz	37% @ ~3BIPS	22% @ ~0.74BIPS	52% @ 1GHz	45.4% @ ~1.2GHz	41% @ 843MHz

(I) Compared to standard 24T DFF

(II) Listed extra # of Transistor includes transistor count in local clock generation (30 Trs, Fig. 2), amortized over average # of latches per cluster and compared to 24T FF. 3 transistors added to each latch making -5 transistors compared to 24T FF.

(III) Only clock overhead compared to standard flip-flop (IV) Transistor counts divided by 4

(V) Iso-voltage comparison, [3,6] Compare PoFF with Margined Baseline; iRazor: Compare Margined Razor with Margined Baseline

(VI) Iso performance comparison

achieves 26-34% performance (power) gain (33-41%) compared to a baseline design across the 0.6 to 1V voltage range. Power reduction becomes 17-26% when comparing to the popular canary approach, at the cost of 13.6% area overhead.

CHAPTER VII

Conclusions and Future Directions

7.1 Contributions

This dissertation focuses on designing energy-efficient secure chips for securing IoT systems. Other design criteria like flexibility, lightweightness and resistance to attacks are also taken into considerations.

Chapter 2 discusses about the cryptography-related mathematics related in the following chip designs. Chapter 3 presents a dedicated AES accelerator, which achieves best-in-class area and energy efficiency by manipulating datapath and optimizing registers. Chapter 4 provides a crypto-coprocessor design to accelerate a wide range of security algorithms. Recryptor proposes a new reconfigurable platform which accelerates cryptographic primitives by replacing the standard SRAM bank with a custom bank using in-memory and near-memory computing elements. To further easily exploring the architecture and analyzing high workload simulations, Chapter 5 desribes a simulator for in-memory computation. Elliptic curve arithmetic and AES-GCM are evaluated. Chapter 6 proposes an error resilient circuit design to tolerate unpredictable variations in operating temperature and voltage, variations in the manufacturing process, and transistor aging of VLSI systems. This adaptive approach against competing industrial techniques such as frequency binning, critical path monitors and canary circuits.

7.2 Future Directions

Building energy-efficient cryptosystems has been the primary focus of my research thus far. However, many of the circuit techniques commonly employed to reduce power and energy consumption actually expose an inherent tradeoff between energy and security: side-channel attacks. For example, power-based side-channel attacks gather information about power consumption of a device over time to deduce the secret key and other protected information, and since IoT devices are physically exposed to attackers, this is an important threat to address. Simply implementing theoretically proven crypto-algorithms is not enough for this domain. The future research would investigate ways to build energy-efficient cryptosystems that are also protected against invasive and noninvasive side channel attacks.

Also, it would be useful to build power sources that can both increase the lifetime of devices and help reduce information leakage through power side-channels. In this vein, new methods of energy-harvesting and modeling of will be very useful for designing secure IoT cryptosystems.

BIBLIOGRAPHY

[1] NIST SP 800-38D. Recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. https://csrc.nist.gov/publications/detail/sp/800-38d/final, Nov 2007.

[2] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das. Compute caches. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 481–492, Feb 2017.

[3] Shaizeen Aga and Satish Narayanasamy. Invisimem: Smart memory defenses for memory bus side channel. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 94–106, New York, NY, USA, 2017. ACM.

[4] Massimo Alioto. *Enabling the Internet of Things: From Integrated Circuits to Integrated Systems*. Springer International Publishing, Cham, 2017.

[5] D. Aranha and C. P. L. Gouvêa. Relic cryptographic toolkit. https://code.google.com/p/relic-toolkit/, 2013.

[6] Diego F. Aranha, Ricardo Dahab, Julio López, and Leonardo B. Oliveira. Efficient implementation of elliptic curve cryptography in wireless sensors. *Advances in Mathematics of Communications*, 4(2):169–187, 2010.

[7] Atmel. Secure memory with authentication at88sc153. http://www.atmel.com/atmel/acrobat/doc1016.pdf.

[8] Atmel. Understanding cryptomemory, the world's only secure serial eeprom. http://www.atmel.com/Images/crypto_und_3_04.pdf.

[9] Josep Balasch, Benedikt Gierlichs, Roel Verdult, Lejla Batina, and Ingrid Verbauwhede. *Power Analysis of Atmel CryptoMemory – Recovering Keys from Secure EEPROMs*, pages 19–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[10] Gordon Bell. Bell's law for the birth and death of computer classes. *Commun. ACM*, 51(1):86–94, January 2008.

[11] D.J. Bernstein. Failures of secret-key cryptography, 2013.

[12] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[13] Joseph Birr-Pixton. Benchmarking modern authenticated encryption on €1 devices. https://jbp.io/2015/06/01/modern-authenticated-encryption-for-1-euro, June 2015.

[14] K. A. Bowman, C. Tokunaga, T. Karnik, V. K. De, and J. W. Tschanz. A 22nm dynamically adaptive clock distribution for voltage droop tolerance. In *2012 Symposium on VLSI Circuits (VLSIC)*, pages 94–95, June 2012.

[15] K. A. Bowman, J. W. Tschanz, N. S. Kim, J. C. Lee, C. B. Wilkerson, S. L. L. Lu, T. Karnik, and V. K. De. Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance. *IEEE Journal of Solid-State Circuits*, 44(1):49–63, Jan 2009.

[16] K. A. Bowman, J. W. Tschanz, S. L. L. Lu, P. A. Aseron, M. M. Khellah, A. Raychowdhury, B. M. Geuskens, C. Tokunaga, C. B. Wilkerson, T. Karnik, and V. K. De. A 45 nm resilient microprocessor core for dynamic variation tolerance. *IEEE Journal of Solid-State Circuits*, 46(1):194–208, Jan 2011.

[17] D. Bull, S. Das, K. Shivashankar, G. S. Dasika, K. Flautner, and D. Blaauw. A power-efficient 32 bit arm processor using timing-error detection and correction for transient-error tolerance and adaptation to pvt variation. *IEEE Journal of Solid-State Circuits*, 46(1):18–31, Jan 2011.

[18] S. Das, D. Roberts, Seokwoo Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. A self-tuning dvs processor using delay-error detection and correction. *IEEE Journal of Solid-State Circuits*, 41(4):792–804, April 2006.

[19] S. Das, C. Tokunaga, S. Pant, W. H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull, and D. T. Blaauw. Razorii: In situ error detection and correction for pvt and ser tolerance. *IEEE Journal of Solid-State Circuits*, 44(1):32–48, Jan 2009.

[20] R. de Clercq, L. Uhsadel, A. Van Herrewege, and I. Verbauwhede. Ultra low-power implementation of ecc on the arm cortex-m0+. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2014.

[21] Q. Dong, S. Jeloka, M. Saligane, Y. Kim, M. Kawaminami, A. Harada, S. Miyoshi, D. Blaauw, and D. Sylvester. A 0.3v vddmin 4+2t sram for searching and in-memory computing using 55nm ddc technology. In *2017 Symposium on VLSI Circuits*, pages C160–C161, June 2017.

[22] A. Drake, R. Senger, H. Deogun, G. Carpenter, S. Ghiasi, T. Nguyen, N. James, M. Floyd, and V. Pokala. A distributed critical-path timing monitor for a 65nm high-performance microprocessor. In *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, pages 398–399, Feb 2007.

[23] D. Ernst, Nam Sung Kim, S. Das, S. Pant, R. Rao, Toan Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pages 7–18, Dec 2003.

[24] Adi Shamir Eyal Ronen, Colin O'Flynn and Achi-Or Weingarten. Iot goes nuclear: Creating a zigbee chain reaction. http://iotworm.eyalro.net/.

[25] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. M. Harris, D. Blaauw, and D. Sylvester. Bubble razor: Eliminating timing margins in an arm cortex-m3 processor in 45 nm cmos using architecturally independent error detection and correction. *IEEE Journal of Solid-State Circuits*, 48(1):66–81, Jan 2013.

[26] P. Franco and E. J. McCluskey. Delay testing of digital circuits by output waveform analysis. In *1991, Proceedings. International Test Conference*, pages 798–, Oct 1991.

[27] M. Peeters G. Bertoni, J. Daemen and G. Van Assche. The keccak sha-3 submission. *Submission to NIST (Round 3)*, 2011.

[28] Flavio D. Garcia, Peter van Rossum, Roel Verdult, and Ronny Wichers Schreur. Dismantling securememory, cryptomemory and cryptorf. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 250–259, New York, NY, USA, 2010. ACM.

[29] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, May 1996.

[30] A. Greenberg. Hackers remotely kill a jeep on the highway – with me in it. Wired, July 2015.

[31] S. Gueron. Intel Advanced Encryption Standard (AES) New Instructions Set, Sept. 2012.

[32] Panu Hamalainen, Timo Alho, Marko Hannikainen, and Timo D. Hamalainen. Design and implementation of low-area and low-power aes encryption hardware core. In *Proceedings of the 9th EUROMICRO Conference on Digital System Design*, DSD '06, pages 577–583, Washington, DC, USA, 2006. IEEE Computer Society.

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

[34] K. Hirairi, Y. Okuma, H. Fuketa, T. Yasufuku, M. Takamiya, M. Nomura, H. Shinohara, and T. Sakurai. 13power supply voltage control with parity-based error prediction and detection (pepd) and fully integrated digital ldo. In *2012 IEEE International Solid-State Circuits Conference*, pages 486–488, Feb 2012.

[35] Michael Hutter, Martin Feldhofer, and Johannes Wolkerstorfer. *A Cryptographic Processor for Low-Resource Devices: Canning ECDSA and AES Like Sardines*, pages 144–159. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[36] Jupiter Instruments. I2c bus monitor. http://www.jupiteri.com/, April 2016.

[37] ISO/IEC. Information technology - security techniques - authenticated encryption. https://www.iso.org/standard/46345.html, 19772:2009.

[38] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw. A 28 nm configurable memory (tcam/bcam/sram) using push-rule 6t bit cell enabling logic-in-memory. *IEEE Journal of Solid-State Circuits*, 51(4):1009–1021, April 2016.

[39] D. Jeon, M. Seok, C. Chakrabarti, D. Blaauw, and D. Sylvester. A super-pipelined energy efficient subthreshold 240 ms/s fft core in 65 nm cmos. *IEEE Journal of Solid-State Circuits*, 47(1):23–34, Jan 2012.

[40] S. Kim, I. Kwon, D. Fick, M. Kim, Y. P. Chen, and D. Sylvester. Razor-lite: A side-channel error-detection register for timing-margin recovery in 45nm soi cmos. In *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 264–265, Feb 2013.

[41] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, Dec 1987.

[42] Kokke. Tiny aes in c. https://github.com/kokke/tiny-AES128-C.

[43] J. W. Lee, S. C. Chung, H. C. Chang, and C. Y. Lee. Processor with side-channel attack resistance. In *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 50–51, Feb 2013.

[44] ARM Holdings Ltd. Arm keil simulator. http://www2.keil.com/mdk5/simulation/.

[45] X. Lu, I. H. Kim, A. Xhafa, J. Zhou, and K. Tsai. Reaching 10-years of battery life for industrial iot wireless sensor networks. In *2017 Symposium on VLSI Circuits*, pages C66–C67, June 2017.

[46] S. Mathew, S. Satpathy, V. Suresh, M. Anders, H. Kaul, A. Agarwal, S. Hsu, G. Chen, and R. Krishnamurthy. 340 mv-1.1 v, 289 gbps/w, 2090-gate nanoaes hardware accelerator with area-optimized encrypt/decrypt gf$(2^4)^2$ polynomials in 22 nm tri-gate cmos. *IEEE Journal of Solid-State Circuits*, 50(4):1048–1058, April 2015.

[47] S. K. Mathew, F. Sheikh, M. Kounavis, S. Gueron, A. Agarwal, S. K. Hsu, H. Kaul, M. A. Anders, and R. K. Krishnamurthy. 53 gbps native $rmGF(2^4)^2$ composite-field aes-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessors. *IEEE Journal of Solid-State Circuits*, 46(4):767–776, April 2011.

[48] C. Miller and C. Valasek. Remote exploitation of an unaltered passenger vehicle. Proceedings of Black Hat 2015, Aug 2015.

[49] Victor S. Miller. *Use of Elliptic Curves in Cryptography*, pages 417–426. Springer Berlin Heidelberg, Berlin, Heidelberg, 1986.

[50] Andrew Moon. the system for unified performance evaluation related to cryptographic operations and primitives(supercop) toolkit. https://github.com/floodyberry/supercop.

[51] J. Myers, A. Savanth, D. Howard, R. Gaddh, P. Prabhat, and D. Flynn. 8.1 an 80nw retention 11.7pj/cycle active subthreshold arm cortex-m0+ subsystem in 65nm cmos for wsn applications. *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, pages 1–3, Feb 2015.

[52] M. Nakai, S. Akui, K. Seno, T. Meguro, T. Seki, T. Kondo, A. Hashiguchi, H. Kawahara, K. Kumano, and M. Shimura. Dynamic voltage and frequency management for a low-power embedded microprocessor. *IEEE Journal of Solid-State Circuits*, 40(1):28–35, Jan 2005.

[53] National Institute of Standards and Technology (NIST). Advanced encryption standard (aes). *FIPS PUBS 197*, 48, Nov 2001.

[54] M. Nicolaidis. Time redundancy based soft-error tolerance to rescue nanometer technologies. In *Proceedings 17th IEEE VLSI Test Symposium (Cat. No.PR00146)*, pages 86–94, 1999.

[55] NIST. Digital signature standard(dss). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf, July 2013.

[56] NIST. Secure hash standard (shs). https://csrc.nist.gov/publications/detail/fips/180/4/final#pubs-abstract-header, August 2015.

[57] K. J. Nowka, G. D. Carpenter, E. W. MacDonald, H. C. Ngo, B. C. Brock, K. I. Ishii, T. Y. Nguyen, and J. L. Burns. A 32-bit powerpc system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling. *IEEE Journal of Solid-State Circuits*, 37(11):1441–1447, Nov 2002.

[58] NVIDIA. Checklist for building a pc that plays hd dvd or blu-ray movies. ftp://download.nvidia.com/downloads/pvzone/ChecklistforBuildingaHDPC.pdf.

[59] Elisabeth Oswald. Dpa book material. http://dpabook.iaik.tugraz.at/onlinematerial/matlabscripts/, Oct 2017.

[60] Pat Pannuto. M-ulator. https://github.com/lab11/M-ulator.

[61] Peter Pessl and Michael Hutter. *Pushing the Limits of SHA-3 Hardware Implementations to Fit on RFID*, pages 126–141. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[62] J. Rabaey. *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall, 1995.

[63] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[64] G. Sayilar and D. Chiou. Cryptoraptor: High throughput reconfigurable cryptographic processor. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 155–161, Nov 2014.

[65] Peter Schwabe, Bo-Yin Yang, and Shang-Yi Yang. Sha-3 on arm11 processors. In *Proceedings of the 5th International Conference on Cryptology in Africa*, AFRICACRYPT'12, pages 324–341, Berlin, Heidelberg, 2012. Springer-Verlag.

[66] J. L. Shin, R. Golla, H. Li, S. Dash, Y. Choi, A. Smith, H. Sathianathan, M. Joshi, H. Park, M. Elgebaly, S. Turullols, S. Kim, R. Masleid, G. K. Konstadinidis, M. J. Doherty, G. Grohoski, and C. McAllister. The next generation 64b sparc core in a t4 soc processor. *IEEE Journal of Solid-State Circuits*, 48(1):82–90, Jan 2013.

[67] L. Sigal, J. D. Warnock, B. W. Curran, Y. H. Chan, P. J. Camporese, M. D. Mayo, W. V. Huott, D. R. Knebel, C. T. Chuang, J. P. Eckhardt, and P. T. Wu. Circuit design techniques for the high-performance cmos ibm s/390 parallel enterprise server g4 microprocessor. *IBM J. Res. Dev.*, 41(4-5):489–503, July 1997.

[68] Dag Spicer. The world's smallest computer. http://www.computerhistory.org/atchm/the-worlds-smallest-computer/, Mar 2015.

[69] STMicorelectronics. Stsafe-a100. http://www.st.com/en/secure-mcus/stsafe-a100.html.

[70] Y. Suzuki, K. Odagawa, and T. Abe. Clocked cmos calculator circuitry. *IEEE Journal of Solid-State Circuits*, 8(6):462–469, Dec 1973.

[71] J. Tschanz, K. Bowman, S. Walstra, M. Agostinelli, T. Karnik, and V. De. Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance. In *2009 Symposium on VLSI Circuits*, pages 112–113, June 2009.

[72] Y. Wang, Y. Shi, C. Wang, and Y. Ha. Fpga-based sha-3 acceleration on a 32-bit processor via instruction set extension. In *2015 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, pages 305–308, June 2015.

[73] Erich Wenger, Martin Feldhofer, and Norbert Felber. *Low-Resource Hardware Design of an Elliptic Curve Processor for Contactless Devices*, pages 92–106. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[74] Wikipiedia. A history of microprocessor transistor count. http://www.wagnercg.com/Portals/0/FunStuff/AHistoryofMicroprocessorTransistorCount.pdf, 2013.

[75] Alexander Wild, Tim Güneysu, and Amir Moradi. *Attacking Atmel's CryptoMemory EEPROM with Special-Purpose Hardware*, pages 389–404. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[76] K. Yang, D. Blaauw, and D. Sylvester. Hardware designs for security in ultra-low-power iot systems: An overview and survey. *IEEE Micro*, 37(6):72–89, November 2017.

[77] Y. Zhang, M. Khayatzadeh, K. Yang, M. Saligane, N. Pinckney, M. Alioto, D. Blaauw, and D. Sylvester. 8.8 irazor: 3-transistor current-based error detection and correction in an arm cortex-r4 processor. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 160–162, Jan 2016.

[78] Y. Zhang, L. Xu, Q. Dong, J. Wang, D. Blaauw, and D. Sylvester. Recryptor: A reconfigurable cryptographic cortex-m0 processor with in-memory and near-memory computing for iot security. *IEEE Journal of Solid-State Circuits*, 53(4):995–1005, April 2018.

[79] Y. Zhang, L. Xu, K. Yang, Q. Dong, S. Jeloka, D. Blaauw, and D. Sylvester. Recryptor: A reconfigurable in-memory cryptographic cortex-m0 processor for iot. In *2017 Symposium on VLSI Circuits*, pages C264–C265, June 2017.

[80] Yiqun Zhang. Recryptor-mulator. https://github.com/novazhang613/M-ulator.

[81] Yiqun Zhang, Kaiyuan Yang, M. Saligane, D. Blaauw, and D. Sylvester. A compact 446 gbps/w aes accelerator for mobile soc and iot in 40nm. In *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, pages 1–2, June 2016.

[82] W. Zhao, Y. Ha, and M. Alioto. Novel self-body-biasing and statistical design for near-threshold circuits with ultra energy-efficient aes as case study. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(8):1390–1401, Aug 2015.