Secure, Resilient and Low-Energy Hardware Architectures for Internet-of-Things

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Sandhya Koteshwara

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTORATE OF PHILOSOPHY

KESHAB K. PARHI

September, 2018

Acknowledgements

First, I would like to thank my advisor Prof. Keshab Parhi for his mentoring, support and advice throughout the course of my Ph.D. His patience in providing me feedback over the years as well as his encouragement for my work has helped me to become a better researcher and successfully complete my thesis.

I would also like to thank Prof. Chris Kim, Prof. Nick Hopper, Prof. John Sartori and Prof. Gerald Sobelman at the University of Minnesota for their support as members of my committee.

My sincere thanks to Dr. Amitabh Das at Intel for his mentorship and collaboration which has resulted in several good research ideas. Special thanks to Prof. Parhi, Prof. Kim and Dr. Das for their recommendations which has helped in my selection at various prestigious forums and future work position.

I thank my current and former labmates Dennis Chu, Bhaskar Sen, Sandeep Avvaru, Xingyi Liu, Nanda Kumar Unnikrishnan, Anoop Koyily, Yin Liu, Ahmad Salehi, Zisheng Zhang, Tingting Xu and Yingjie Lao.

I am also grateful to my mother, father, sister, brother-in-law and Tanshu (Tinkoo) for their unconditional love, support and belief in me and my decisions.

Finally, I would like to thank my friends over the years of my Ph.D. for their amazing company, time, love and encouragement. Special thanks to Nandini, Pooja, Surya, Azra, Bala, Gattu, Shreyas, KK, Ravi, Maya, Anand, Supriya, Bhaskar, Sandeep, Anoop, Sacchiddh, Pratibha, Kane, Alex, Purti, Panda, Prakash, Anagha, Biswa, Sampreeti, David and Krishna. This Ph.D. would not have been possible without them.

Dedication

To the determination, perseverance and incredible strength of the human spirit!

Abstract

Modern devices are not only shrinking in size but also being increasingly connected to each other. These smart, connected devices are broadly termed *Internet of Things* or IoT. Examples of IoT applications range from safety critical automotive IoT employed in self-driving cars, smart cards and radio-frequency identification (RFIDs), high speed wireless sensor nodes, consumer electronics such as smart TVs, refrigerators and critical, life-saving systems such as implantables and medical devices. The devices used in these applications demand low cost, high efficiency and high security algorithms to be implemented using the most resource efficient platforms. To keep up with the changing application scenario, new architectural mappings and techniques for designs beyond traditional optimization has become a necessity.

This thesis focuses on development of hardware architectures to achieve high end-to-end security and low cost, low energy solutions using both Application-Specific Integrated Circuits (ASIC) designs and Field Programmable Gate Array (FPGA) implementations. The proposed architectures are targeted at ensuring security in manufacturing of semiconductor circuits, resilient communication between devices in safety critical systems, and low-energy solutions for machine learning applications.

First, hardware obfuscation architectures for secure manufacturing are proposed. Hardware security has emerged as an important topic over the last decade due to reports of counterfeit devices detected in safety-critical defense systems. With the upcoming era of IoT, security is only going to increase in importance with more devices being manufactured and connected to a common network. Hardware obfuscation, which is defined as hiding functionality of designs and locking them using secret keys before sending for manufacturing, attempts to improve security of circuits. This technique ensures that the circuit is not understandable or usable by parties not possessing the correct secret key. Once the chips are received from the untrusted manufacturing units, they are unlocked and distributed to the end user, ensuring reliability. Development of novel architectures and schemes for obfuscation is the first topic of my research.

Second, Authenticated Encryption architectures for resilient communication are presented. Authenticated encryption with Associated Data (AEAD) is a form of encryption

which simultaneously provides confidentiality, integrity, and authenticity assurances on the data. Authentication verifies that the source of information is a trusted device and encryption ensures that the message has been protected from modifications and eavesdropping across the channel. To ensure secure communication between two devices connected on the network, low cost and low power AEAD schemes are of paramount importance. To address design of authenticated encryption algorithms tailored towards changing application scenarios, a Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) is currently in progress. Mapping of algorithms from CAESAR to low power and energy architectures using FPGA is my second topic of research.

Finally, low energy machine learning architectures for biomedical internet-of-things are proposed. Biomedical applications such as implantable devices require low power and energy efficient architectures to perform machine learning tasks. For example, seizure detection aims to classify between resting and seizure state of electroencephalogram (EEG) signals. To differentiate between the two states, distinguishing features need to be extracted from these signals and classifiers which are trained on large amounts of training data need to be employed. The units required to extract features involve power spectral density (PSD) computation and support vector machine (SVM) classifiers, which require high energy. A technique termed approximate computing is applied to decrease the energy consumption of systems. Specifically, applying bit-width reduction to modify architectures to lower energy consumption by incrementally increasing the precision in stages and using multi-level classifiers while maintaining high sensitivity and specificity is the third focus of my research.

Contents

Acknowledgements i

Dedication ii

Abstract iii

List of Tables vi

List of Figures vii

1 Introduction 1

1.1 Overview 1

1.2 Contributions 2

1.3 Outline of Thesis 4

2 Mode-based obfuscation 5

2.1 Introduction 5

2.2 Obfuscation Methodology 7

2.2.1 Folding transformation on FFT circuits 7

2.2.2 Control-flow changes for obfuscation 8

2.3 Complete design 11

2.3.1 Design of meaningful modes 11

2.3.2 Design of non-meaningful modes 12

2.4 Analysis of obfuscated modes 13

2.4.1 Assumptions of the attack model 15

2.4.2 Obscurity of control-flow 15
2.4.3 Protection of length of FFT 16
2.5 Area and power overhead 17
2.6 Conclusion 19

3 Dynamic obfuscation **20**
3.1 Introduction 21
3.2 Related work 22
3.3 Fixed, Time-Varying and Dynamic key-based obfuscation 24
3.3.1 Basic Concepts of Key-Based Obfuscation 24
3.3.2 Mode based Control-flow modification 26
3.3.3 Example of mode-based obfuscation 27
3.3.4 Time-varying and Dynamic sequentially-triggered modes 29
3.4 Trigger circuits and trigger combination circuits 30
3.4.1 Design of trigger circuits 30
3.4.2 Design of trigger combination circuits 36
3.5 Security Analysis 38
3.5.1 Lower bounds on security using brute-force attack 39
3.5.2 Average time to attack using brute-force 40
3.5.3 Average time to attack using Reverse engineering 42
3.5.4 SAT solver based attacks 44
3.6 Results 45
3.6.1 Overhead of trigger circuits 45
3.6.2 Overhead of sequentially-triggered mode-based obfuscation and comparison with fixed-mode obfuscation 46
3.7 Algorithm 49
3.8 Dynamic obfuscation using two random number generators 50
3.9 Dynamic obfuscation using true random number generators 51
3.10 Conclusions 55
3.11 Proof of average percentage of incorrect outputs per key 56
3.12 Proof of average time to attack using brute force 57

4 Comparative study of Authenticated Encryption **59**
4.1 Introduction . . . 59
4.2 AES-GCM . . . 61
4.3 CAESAR competition . . . 62
4.3.1 Primitives . . . 63
4.3.2 Features . . . 63
4.4 Performance analysis . . . 66
4.4.1 Software implementations . . . 66
4.4.2 Hardware implementations . . . 66
4.4.3 Memory footprint . . . 68
4.5 Summary with recommendations of candidates . . . 68
4.5.1 Block-cipher based . . . 68
4.5.2 Sponge/Duplex based . . . 71
4.5.3 Others . . . 71
4.6 Conclusions and future work . . . 72

5 Nonce-Misuse Resistant Authenticated Encryption Architectures **73**
5.1 Introduction . . . 74
5.2 Motivation and Related work . . . 75
5.2.1 New AEAD schemes . . . 76
5.2.2 Importance of nonce-misuse resistance . . . 77
5.2.3 Description of AES-GCM . . . 78
5.2.4 Architecture of AES-GCM . . . 78
5.3 Algorithms with nonce-misuse resistance . . . 80
5.3.1 Deoxys . . . 80
5.3.2 AES-GCM-SIV . . . 81
5.3.3 POET . . . 83
5.3.4 PRIMATE-APE . . . 84
5.4 Architectural descriptions . . . 85
5.4.1 Deoxys . . . 86
5.4.2 AES-GCM-SIV . . . 86
5.4.3 POET . . . 87

5.4.4 PRIMATE-APE . . . 88
5.5 Optimizations . . . 90
5.5.1 Parallel processing of messages . . . 90
5.5.2 Implementation of S-Box in memory . . . 92
5.5.3 Tower field implementation of block cipher . . . 92
5.6 Hardware-Software co-design . . . 93
5.6.1 Codesign of AES-GCM . . . 93
5.6.2 Optimization of the co-design . . . 94
5.6.3 Application of co-design techniques to AES-GCM-SIV and Deoxys 94
5.7 Results . . . 97
5.7.1 Experimental setup . . . 97
5.7.2 Implementation 1: Direct implementation of the algorithms using combinational logic only . . . 99
5.7.3 Implementation 2: Optimization using parallel processing . . . 100
5.7.4 Implementation 3: Optimization of the S-Box of the block ciphers using memory blocks . . . 101
5.7.5 Implementation 4: Optimization using tower field implementations 102
5.7.6 Implementation 5: Optimization using hardware-software co-design approach . . . 103
5.8 Analysis of side-channel attacks and Countermeasures . . . 104
5.9 Recommendations and Conclusions . . . 105

6 Low-Energy Architectures of Linear Classifiers 107
6.1 Introduction . . . 107
6.2 Incremental-precision Addition and Subtraction . . . 108
6.3 Incremental-precision Fixed-Width multipliers . . . 109
6.3.1 Fixed-width multipliers . . . 110
6.3.2 Approximate carry generation . . . 111
6.3.3 Two-stage incremental-precision multipliers . . . 113
6.3.4 Multi-stage incremental-precision multipliers . . . 114
6.4 Multi-level classification algorithm . . . 116
6.5 Experimental results . . . 117

6.5.1 Multi-level linear classification . . . 117
6.5.2 Accuracy of classification . . . 120
6.5.3 Test accuracy and reduction in energy using incremental-precision components . . . 120
6.6 Conclusions . . . 121

7 Incremental-Precision based Feature Computation and Multi-Level Classification 122
7.1 Introduction . . . 123
7.2 Related work . . . 124
7.3 Incremental-precision feature extraction and classification system . . . 126
7.3.1 Training steps . . . 126
7.3.2 Threshold calculation . . . 127
7.3.3 Testing steps . . . 130
7.3.4 Similarities and differences between AdaBoost and incremental precision algorithm . . . 131
7.4 Incremental-Precision Hardware Architectures using Data-Path Decomposition . . . 132
7.4.1 Decomposition of adders and multipliers . . . 132
7.4.2 Decomposition of data-path of FFT to form incremental-precision architectures . . . 133
7.5 Error and Energy requirement analysis . . . 135
7.5.1 Modeling of error due to input approximation and approximate multipliers . . . 136
7.5.2 Resource consumption and overhead for incremental-precision architectures . . . 139
7.6 Case study: Seizure detection using EEG signals . . . 141
7.6.1 Dataset and System description . . . 141
7.6.2 Experimental setup . . . 142
7.7 Results . . . 144
7.7.1 Training accuracy . . . 144
7.7.2 Testing accuracy and Power reduction . . . 145

7.7.3 Overhead of selected configurations 149
7.8 Conclusions and Future work 150
7.9 Appendix A 150
7.10 Appendix B 153

8 Conclusion and Future Work **159**

References **161**

List of Tables

2.1 Overhead due to meaningful modes 17
2.2 Overhead due to size of mux at controlpath 18
2.3 Overhead for different key sizes 18
3.1 Trigger period, duty cycle and percentage incorrect outputs for different key sizes and L=100 36
3.2 Brute-force attack time for different key values, trigger period and L with system operating at 100MHz 41
3.3 Time complexity of reverse engineering attack at multiplexers for different key sizes and values of L 43
3.4 Gate counts of counter based trigger circuits for different key sizes . . . 45
3.5 Comparison of the overhead and time to attack of dynamic and fixed obfuscation for various sequential circuits with different activation rates L 47
3.6 Trigger period, duty cycle and percentage incorrect outputs for different key sizes and $size\_trng = K$ 54
3.7 Brute-force attack time and reverse engineering attack time for different key sizes K 55
4.1 Comparison of candidates based on nonce misuse resistance, security and parallelizability 65
4.2 Candidates with implementations on embedded platforms and associated memory footprints 70
4.3 Summary of candidates highlighting properties and possible application scenarios 71
5.1 Summary of candidate features and comparison with AES-GCM 81
5.2 Table of opcodes and their corresponding operations for decoders 95

5.3 Area, throughput and power of AES-GCM and other algorithms using direct implementation (Frequency = 50 MHz) 100
5.4 Area, throughput and power of AES-GCM and other algorithms using parallel processing of messages (Frequency = 50 MHz) 101
5.5 Area, Throughput, Power and Memory requirements using optimization of S-Box using memory blocks (Frequency = 50MHz) 102
5.6 Results of optimization using tower field implementations of AES 103
5.7 Results of hardware/software co-design implementation 104
5.8 Recommendations of candidates based on observed results 105
6.1 Table of partial products and the expected values of partial products for every combination of recoded bits y' and input $x_j x_{j-1}$ 112
6.2 Accuracy of classification for the training and testing process using full-precision single stage and incremental-precision multi-level classifier . . 119
6.3 Accuracy of the classifier based on implementations of the full-precision and multi-level algorithms 119
6.4 Area, Power, Timing and Energy consumption per test sample for the single stage full-precision and multi-level incremental precision algorithms (Numbers in paranthesis indicate values after recalculation of threshold) 119
7.1 Hardware complexity of a 1024-point FFT in terms of real adders, real multipliers and real delay elements. Cells with two values indicate the resource count and corresponding bit precision in parentheses. The multipliers correspond to 4 stages and the adders correspond to 10 stages. . 156
7.2 Estimates of area, power consumption and timing of critical path of the 1024-point incremental-precision architecture compared to a 16-bit precision architecture. The resource consumption values and performance numbers are obtained using synthesis of components with Design Compiler, 65nM technology library and a clock frequency of 100MHz 156
7.3 Performance of selected configurations with respect to final training accuracy, specificity, sensitivity, power reduction, area reduction, timing overheads and energy consumption 157

7.4 Comparison of testing accuracy and energy consumption of proposed incremental-precision system applied to seizure detection with similar approaches from literature 158

List of Figures

2.1 Design flow of mode-based obfuscation using Control-flow modifications 7
2.2 16 point Radix-2^2 complex DIF FFT flowgraph 8
2.3 16-point, 2-parallel folded FFT architecture 8
2.4 Major blocks of the folded FFT architecture 9
2.5 Modification of control logic of delay-switch-delay structure (input s) . . 10
2.6 Effect of modification of -j multiplication factors on output 10
2.7 Modification of control logic of BFII (input t) 11
2.8 256-point, 2-parallel folded FFT architecture 12
2.9 Obfuscated FFT with four meaningful modes built on 1024-point, 2-parallel folded FFT 12
2.10 Obfuscated control-path with various signals derived from a counter . . 14
3.1 Comparison of encryption scheme as defined in cryptography with hardware obfuscation technique for protection during manufacturing 22
3.2 Difference between fixed, time-varying and dynamic obfuscation using multiplexers (Correct key values are $K[0] = 0$ and $K[1] = 1$) 25
3.3 High level overview of mode-based obfuscation using control-flow modifications using multiplexers 26
3.4 Datapath, components and required control signals of a Folded, Radix-2^2 1024-point FFT 27
3.5 Incorrect outputs due to modification of control s of switches 28
3.6 Corruption of outputs due to modification of control t of butterfly units 28
3.7 Deriving correct and modified control signals from counter and mapping of key bits 28

3.8 Trigger circuit combined with mode-based obfuscation to create time-varying and dynamic obfuscation 29
3.9 Different types of hardware trojan circuits based on the method of obtaining trigger signals 31
3.10 Number of multiplexers and Trigger signals required to map a key size of K bits 32
3.11 Structure of a sequential trigger generation circuit which uses a series of counters (cntd, cntf) and a random number generator output (rng) to generate triggers $T1$ and $T2$ 33
3.12 Timing diagram of trigger signals $T1$ and $T2$ based on conditions used in the trigger generation circuit 35
3.13 Logic gates of the trigger combination circuit to combine the generated trigger signals with existing control signals 37
3.14 Trigger combination circuit added between derived control signals and the key-bit mapped multiplexers for a key size = 8 bits 38
3.15 Timing diagram of corruption of outputs in fixed obfuscation and trigger signals in time-varying obfuscation 40
3.16 Structure of a sequential trigger generation circuit which uses two random number generators (rng1 and rng2) to generate triggers $T1$ and $T2$ . . . 50
3.17 Structure of an asynchronous trigger generation circuit which uses a true random number generator (TRNG) to generate triggers $T1$ and $T2$ . . . 52
4.1 General scheme of an Authenticated Encryption scheme with Associated Data 60
4.2 Simplified block diagram of AES-GCM using minimal resources 62
4.3 Software performance numbers based on candidate implementations and SUPERCOP benchmarks 67
4.4 Hardware performance numbers based on candidate implementations and ATHENa tools 69
5.1 Description of the algorithm of AES-GCM. The left part of the figure presents ciphertext block generation using a counter. The right part of the figure presents associated data processing and processing of plaintext blocks to generate the tag. 79

5.2 Data-path of AES-GCM consisting of AES block and Galois Field multiplier block. 79
5.3 Finite State Machines (FSM) of AES-GCM for encryption and authentication in a block-interleaved manner. Signal ct_done is used for synchronization between the two FSMs. 80
5.4 Description of the algorithm of Deoxys. The encryption block in this figure is the tweakable Deoxys block cipher with key and tweak as inputs. The top part of the figure presents tag generation. The generated tag is used for processing of plaintext blocks to generate ciphertext blocks. . . 82
5.5 Algorithm description of AES-GCM-SIV. The left part of the figure represents processing of associated data and message blocks using the POLYVAL and AES block to produce a Tag. The right half of the figure represents ciphertext generation using the Tag as an initial value for the counter. 83
5.6 Algorithm description of POET. The left part of the figure presents processing of associated data/header blocks to produce an intermediate value τ. Ciphertext and Tag generation using the intermediate τ value are presented in the right half of this figure. 84
5.7 Algorithm description of PRIMATE-APE. After initial absorption of the nonce and associated data by the PRIMATE block, for every message block that is input, the cipher text block is produced. Finally, the tag is generated. 85
5.8 Datapath and Controlpath of the Deoxys algorithm. Note that the block cipher is a tweakable block cipher and a single state machine is used for both authentication and encryption. 87
5.9 Data-path and control-path of AES-GCM-SIV. Only the conversion of *GF_MULT* of AES-GCM to *POLYVAL* of AES-GCM-SIV is presented here. Also note that a single FSM is used for both authentication and encryption. 88
5.10 Datapath and Controlpath of the POET algorithm. Note that the datapath consists of a regular AES 10 round cipher and two AES 4 round ciphers used as hash function blocks. 89

5.11 Datapath and Controlpath of PRIMATE-APE algorithm. Note that the implementation of the PRIMATE block is similar to AES block but is much smaller and utilizes nibbles for processing. 90
5.12 Parallel processing of messages in AES-GCM-SIV. 91
5.13 Parallel processing of messages in Deoxys. 92
5.14 Block diagram of a co-design implementation using NIOS II soft processor. The top part of the figure represents co-design on a AES-GCM/AES-GCM-SIV module. The bottom part presents co-design on Deoxys. . . . 96
5.15 Experimental setup of the Atlas SoC board interfaced with a PC using the System Console. The board hosts an Altera Cyclone V FPGA (5CSEMA4U23C6). 97
6.1 Decomposition and subsequent combination operations on 8-bit inputs to create incremental-precision adder/subtraction unit. The 8-bit inputs are decomposed into two 4-bit inputs in this example. 109
6.2 Data-path decomposition of multiplier into multiple stages by incrementally increasing the precision of x and maintaining y at a constant precision. 110
6.3 Multiplication of two 8-bit inputs by applying Booth recoding on the input y. The resultant output is of length $2W - 1$ bits. 111
6.4 Calculation of the error compensation bias terms LP_major and LP_minor based on carry signals. 112
6.5 Splitting the 8-bit multiplier into two approximate multipliers of 4 bits each. 113
6.6 Splitting the LSB 4-bit multiplier into two approximate multipliers of 2 bits each. 115
6.7 Incremental-precision 8-bit multiplier using one 4-bit multiplier and one time-multiplexed 2-bit multiplier. Intermediate outputs are stored in memory. 115
6.8 Steps of the multi-level classification algorithm with increasing precision classifier at each stage. 116
6.9 Classification of samples using multi-level classification algorithm. . . . 118
7.1 Steps of training process of the incremental-precision algorithm using varying precision features. 128

7.2 Plot of sorted probability estimates of classified data samples for class 0 and class 1 samples. Initial values of thresholds can be set using this data. 129
7.3 Steps of the testing process of incremental-precision algorithm using varying precision features. 130
7.4 16-point, 2-parallel, folded, radix-2^2 real FFT architecture using a hybrid data-path. The components and paths marked in blue are complex data-paths. 134
7.5 Decomposition of 16-point, 2-parallel architecture into upper and lower data-paths with growing bit-widths at each stage. The components and paths marked in blue are complex data-paths. Note that varying precision multipliers are required. The controlpath is common for both the data-paths. 135
7.6 Modeling of error in 16-bit datapath of a 16-point radix-2^2 FFT. Note that the butterfly structures are modeled by a gain of 1/2 followed by rounding noise while the multipliers are modeled using an additional noise source at every alternate stage. 137
7.7 Modeling of error in incremental precision architectures due to approximation. The noise due to addition operation V_{rnd} is included in the total error but not shown. The noise sources due to approximate multiplication propagate to the outputs and are added to the total error after combination of data-paths. 138
7.8 Logarithm of the variance of error for different starting precision, step size and final precision for a 1024-point, 2-parallel, radix-2^2 FFT architecture. 138
7.9 Identification of critical path of a 1024-point 16-bit precision FFT and comparison with the critical path of a 1024-point incremental-precision FFT starting at 4 bits. Note that stages 7 and 8 form part of the critical path for incremental- precision architectures due to a growing data-path. 140
7.10 Low complexity implementation of feature extraction unit for computation of spectral powers. The Power Spectral density can be implemented using a Welch PSD estimate whose main component is an FFT. 142
7.11 Effect of using varying precision features on training accuracy (reported as F1-score) and testing accuracy (reported as Specificity and Sensitivity). 145

7.12 Improvement in training accuracy at different stages of the incremental-precision algorithm for various configurations of M values. The x-axis represents different configurations ranging from 1 to 36 with varying maximum allowed misclassification errors $M1$, $M2$, $M3$ and $M4$ (termed as M values). 146
7.13 Final training accuracy for different starting precision, step size, maximum allowed stages and configurations of M value. 147
7.14 Ratio of power consumption *vs* Sensitivity of test data for different starting precision, step size, maximum stages and configurations of M value. Note that configurations towards the right bottom corner of these plots have high sensitivity and low power ratios. 148
7.15 Data-path decomposition of fixed-width adder operating on signed numbers x_1 and x_2 and subsequent recombination to produce y. The block $[0||x]$ indicates the operation of appending x with one leading zero. . . . 151
7.16 Data-path decomposition of fixed-width multiplier operating on x and coefficient W and subsequent recombinaton to produce approximate output $\hat{y}$. 151
7.17 Architecture of an 8×8 fixed-width multiplier based on modified Booth multiplication and carry generation. 152
7.18 Splitting of an 8×8 bit multiplier into two 4×8 bit multipliers by approximation. The carry information from the top multiplier is merged with the bottom multiplier during recombination. 153
7.19 Data-path decomposition of the butterfly unit which is composed of addition, subtraction and shift operations. Recombination of outputs is deferred to the last stage. 154
7.20 Data-path decomposition of the complex multiplication unit composed of four real multiplications, addition and subtraction operations. 154

Chapter 1

Introduction

1.1 Overview

Billions of devices will be connected to a common network termed the Internet-of-Things (IoT). With more and more devices being connected, there is a shift in the trend of computing towards the edge of the network rather than at the cloud. This is termed as *edge-centric computing*. Edge-centric computing will become prevalent with increased data availability, better computational resources and security and privacy concerns [1,2]. However, this trend will give rise to several challenges which need to be addressed. Some of the challenges of an IoT era are as follows:

- With an increase in the number of devices being produced and connected, there will be an increase in the demand for manufacturing of small integrated circuits and embedded chips.
- As more companies move away from owning and operating an in house foundry, there will an increased demand for outsourcing of fabrication.
- The IoT framework thrives on communication between devices connected at the edge as well as communication between the device and the cloud. These communications demand high security and privacy guarantees.
- The devices at the edge of the network can vary from sophisticated laptops and cellphones to small, low cost smart cards and chips. Hence, the device resources

in many applications could be minimal and not include high quality resources like random number generators which is critical for several cryptographic algorithms.

- Because of the large number of devices connected to the network, tasks such as secret key distribution, update and management for cryptographic algorithms can be a challenging task.

- There will be an increased necessity to run machine learning algorithms such as Support Vector Machines, Neural Networks etc., on the device itself. In many cases it is also desirable to run the training algorithm on the hardware. This requires high computation and memory.

- Since IoT devices are mostly powered by small portable batteries or wireless power transfer, low energy consumption is a must.

In this thesis, we attempt to alleviate some of these challenges by developing hardware architectures aimed at secure, resilient and low energy solutions.

1.2 Contributions

The contributions of this research are as follows:

1. To alleviate the challenge of secure manufacturing, hardware obfuscation which involves modifications to integrated circuits and locking using keys is proposed. My research work presents a novel technique to hardware obfuscation using modifications to control signals of an architecture [3] and extends the technique for application in a hierarchical manner to a complete integrated circuit [4]. Using ASIC implementations and a case study on fast Fourier transform (FFT) architectures, it is demonstrated that the proposed technique is a simple solution requiring changes only to the control flow of the design resulting in low overheads, compared to existing techniques which require major modifications to the design flow.

2. With increased availability of resources and low cost equipment, attacks on obfuscated circuits have grown stronger and current methods of obfuscation are not secure enough and can be easily broken. To address this concern, modifications

are introduced to the mode-based technique of obfuscation to create a dynamic method of obfuscation [5] where the modes are unpredictable and random to increase security of the design by many orders of magnitude. Using FFT circuits, three different attack strategies are applied and it is proved that the dynamic technique has exponentially higher time-to-attack compared to existing methods. A combined hybrid obfuscation technique which presents a trade-off between fixed and dynamic obfuscation is also proposed [6].

3. For resilient communication between devices authenticated encryption algorithms which combine both authentication of source and confidentiality of messages is a must. A competiiton termed Competiiton for Authenticated Encryption (CAESAR): Security, Applicability and Robustness has been proposed to identify new algorithms which are significantly better than current standards. In my research, I have analyzed architecture design of candidates selected to the second round of CAESAR competition with respect to security level, cost and energy consumption. Proposed mappings of these algorithms to different IoT applications, based on their hardware, software and security performance numbers in comparison to an existing standard, AES-GCM [7] are also presented. An overview of this kind does not exist in current literature and is highly beneficial for designers of cryptographic protocols for embedded system platforms and SoCs.

4. Current authenticated encryption algorithms are critically dependent on a value termed nonce for their security guarentees. These nonces rely on good quality random number generators and frequent updates which can be a challenge in IoT systems. In my research, I have identified algorithms which provide nocemisuse resistant property from CAESAR and other proposals. I have completely designed and implemented two architectures for candidates Deoxys [8] from the CAESAR competition and AES-GCM-SIV [9] on Altera Cyclone V FPGA, and performed an area, power and energy analysis with respect to the architectural implementation of current standard, AES-GCM on the same platform. I have also worked on the design and implementation of authenticated encryption algorithm using a hardware/software co-design approach to obtain better resource efficiency on modern application platforms. The novelty of this work is in understanding

trade-off between hardware and software partitioning of the AEAD algorithms [10].

5. My research work proposes a novel theory for classification of samples termed incremental-precision classification [11,12], inspired by an existing boosting theory popular in machine learning applications. This theory employs multiple classification stages using weak classifiers to ultimately produce a strong classification output. This reduces the complexity of systems which can now be built using simpler classifiers. Similar to the boosting approach, the incremental-precision classification approach results in a strong classifier using multiple stages of low-precision classification stages. The resulting multi-stage classifier is proven to reduce energy consumption resulting in systems that can be used in low energy applications.

1.3 Outline of Thesis

The rest of the thesis is organized as follows.

Chapter 1 presents the mode-based obfuscation for hardware obfuscation using modification to control signals of the design.

In Chapter 2, dynamic obfuscation technique is presented in which an incorrect key results in erratic behavior of the circuit in a dynamic manner.

The study of authenticated encryption algorithms based on candidates from the CAESAR competition is presented in Chapter 3.

The detailed architecture of nonce-misuse resistant authenticated algorithms selected from CAESAR competition and alternate proposals is presented in Chapter 4 along with implementations on FPGA and ASIC platforms.

Chapter 5 introduces the concept of incremental-precision classification by demonstrating the technique on adders and multipliers as applicable to linear classifiers.

In Chapter 6, incremental-precision architectures are applied to a seizure detection application and the energy-accuracy trade-off is presented.

Chapter 2

Mode-based obfuscation

Hardware security has emerged as an important topic in the wake of increasing threats on integrated circuits which include reverse engineering, intellectual property (IP) piracy and overbuilding. This chapter explores obfuscation of circuits as a hardware security measure and specifically targets digital signal processing (DSP) circuits which are part of most modern systems. The idea of using desired and undesired *modes* to design obfuscated DSP functions is illustrated using the fast Fourier transform (FFT) as an example. The selection of a mode is dependent on a *key* input to the circuit. The system is said to work in its desired mode of operation only if the correct key is applied. Other undesired modes are built into the design to confuse an adversary. The approach to obfuscating the design involves *control-flow* modifications which alter the computations from the desired mode. We present simulation and synthesis results on a reconfigurable, 2-parallel FFT and discuss the security of this approach. It is shown that the proposed approach results in a *reconfigurable* and flexible design at an area overhead of 8% and a power overhead of 10%.

2.1 Introduction

As more design houses are moving away from fabrication, the supply chain for semiconductor integrated circuits is getting more distributed. This has made control over the quality and security at different levels of manufacturing difficult. Attacks on hardware such as hardware trojans, intellectual property (IP) piracy, overbuilding of integrated

circuits (IC), reverse engineering, side channel attacks and counterfeiting can alter device functionality, lead to loss of jobs and even catastrophic situations including loss of human lives [13]; this has led to a high level of importance on hardware security. Hardware obfuscation, which involves hiding the functionality of a design, helps in protection against many of the above mentioned attacks. Therefore, we address the task of hardware obfuscation in this work.

Several approaches to hardware obfuscation have been proposed in literature including passive methods such as change of code written in a hardware description language (HDL) [14], active methods such as insertion of gates at register transfer level (RTL) [15], combinational logic modifications [16] and netlist-level obfuscation [17]. However, none of the above methods specifically addresses the special challenge of obfuscation of DSP circuits. An approach to hardware obfuscation using high-level transformations has been proposed in [18]. An implementation of this obfuscation technique using counter resetting on a Radix-2 real FFT was shown in [19]. However, no systematic approach to control-flow modifications was presented in this work.

The motivation behind obfuscating DSP circuits is two fold. First, DSP circuits are highly control driven and hence obscuring the control-flow of such designs is critical. Also, DSP circuits are defined by specific properties such as number of taps in a filter, length of the FFT etc. which dictate important constraints such as performance, area and power. Hence, hiding this type of information is another goal of obfuscation. The approach to obfuscation proposed in [18] and used here consists of using a mode-based method where the entire design can operate in meaningful and non-meaningful modes. A *key* to the system decides its mode of operation by selecting the configuration of the datapath and control-flow output from the controlpath. A block diagram to illustrate the approach is shown in Figure 2.1. Additional modes that leak partially incorrect information are also added to aid obfuscation.

The obfuscation scheme protects the circuit in two ways. Locking the entire design using keys prevents illegal piracy and overproduction, since the keys are kept secret with the design house and only programmed into the chip after fabrication. The foundry has no access to the key, rendering the circuit useless even if multiple copies are made. The scheme also achieves a second objective by making the chips harder to reverse engineer. The control-flow of the design is well hidden and concept of multiple modes in the design

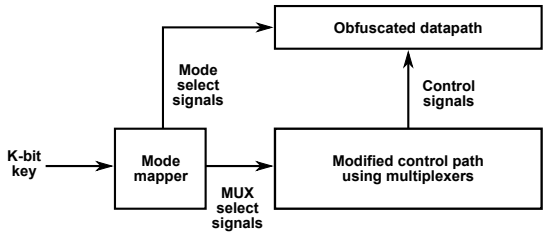


Figure 2.1: Design flow of mode-based obfuscation using Control-flow modifications

prevents competitors who are trying to make use of the design aspects to lower the cost or power of the circuits.

This work presents an obfuscated design of an FFT that can be configured as a 16/64/256/1024-point FFT working in 2-parallel mode. These functional modes are the meaningful modes of the design. We create non-meaningful modes and the method of selection of these modes is addressed in this work. A main contribution of this work is a novel method of modifying the control-flow for obfuscation with low overhead. This work also introduces the notion of modes that compute partially correct outputs. Security analysis of the design with respect to the contribution of the designed meaningful and non-meaningful modes is presented highlighting the importance of this scheme in securing DSP circuits.

2.2 Obfuscation Methodology

In this section, we discuss the basic techniques used to create a mode-based obfuscated design. These methods exploit popularly used transformations and modifiable properties of DSP circuits to affect the outputs in different ways. We use a simple FFT circuit to demonstrate these ideas.

2.2.1 Folding transformation on FFT circuits

Time-multiplexed architectures can be designed by a high-level transformation referred to as *folding* [20]. For a folding factor N, N algorithm operations can be executed by the same hardware functional units in N clock cycles. The inputs to the functional units in a folded or time-multiplexed circuit are selected by control signals. Altering

these control signals alters the functionality of the circuit.

Pipelined, parallel complex FFT architectures can be designed using folding as described in [21]. It is seen that for an FFT of size N, a folding factor of $N/2$ leads to a 2-parallel architecture. To illustrate folding, we consider a 16-point, radix-2^2, decimation-in-frequency (DIF) complex FFT circuit shown in Figure 2.2. Appropriate

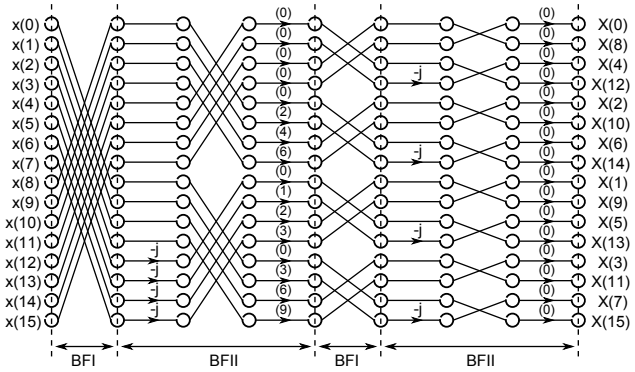


Figure 2.2: 16 point Radix-2^2 complex DIF FFT flowgraph

selection of folding sets results in a 2-parallel structure for this FFT as shown in Figure 2.3. The folded architecture consists of major blocks such as two-point butterfly

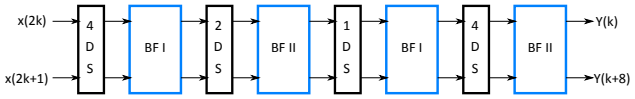
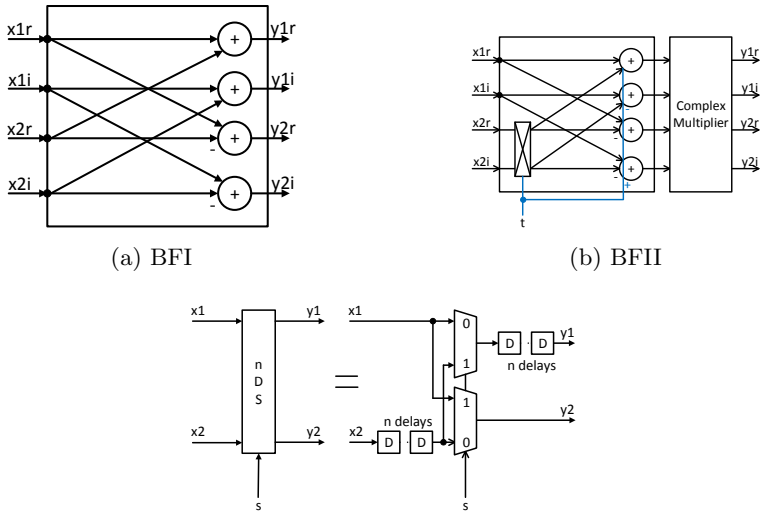


Figure 2.3: 16-point, 2-parallel folded FFT architecture

(BFI), two-point butterfly which also performs trivial multiplication (by -j) and twiddle factor multiplication (BFII), and delay-switch-delay of size n (nDS) structures which are shown in Figure 2.4. Reconfigurability of the design can be achieved by selecting various folding sets and generating structures of 2-parallel operation. We term these folding sets to be *meaningful modes* of the design and a combination of these will be used to create a complete obfuscated system.

2.2.2 Control-flow changes for obfuscation

After applying folding and obtaining appropriate folded FFT architectures, we can exploit the properties of control flow of folded FFT architectures to generate functionally



(a) BFI

(b) BFII

(c) Delay-switch-delay

Figure 2.4: Major blocks of the folded FFT architecture

incorrect modes. We term these *non-meaningful modes*, and show how modifications to signals generated by the control path of the architecture could be used to create them.

For the folded FFT components in Figure 2.4, the control inputs t (which corresponds to multiplication by -j) of BFII and s of switches are specified by the control path. The delay-switch-delay structures operate correctly only if the appropriate control signals derived from a $log_2 N$-bit counter are used. Incorrect patterns applied to these delay-switch-delay structures produce modes which generate random outputs. The randomness of the outputs increases with increase in the number of delay-switch-delay structures with wrong control signals. The correct operation of these structures and the effect of the changes are illustrated in Figure 2.5.

Next, from the flowgraph of Figure 2.2, it is observed that a radix-2^2 FFT has alternate columns of twiddle factors and multiplication by -j terms. In the folded architecture, these changes can be embedded in the BF II structure shown in Figure 2.4. From the folding sets chosen, the scheduling order of each stage can be found. Using this information, we can identify that changes to twiddle factors or -j multiplication will only alter a subset of the outputs. This is an important observation as it leads to generation of non-meaningful modes with partially correct outputs. Such modes provide

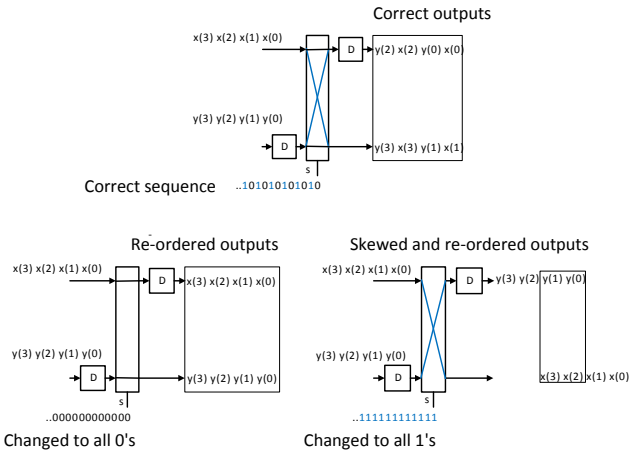


Figure 2.5: Modification of control logic of delay-switch-delay structure (input s)

partial information about the functionality of the system. The role of these modes will be discussed in later sections.

As an example, consider the FFT flowgraph shown in Figure 2.6. Suppose the sys-

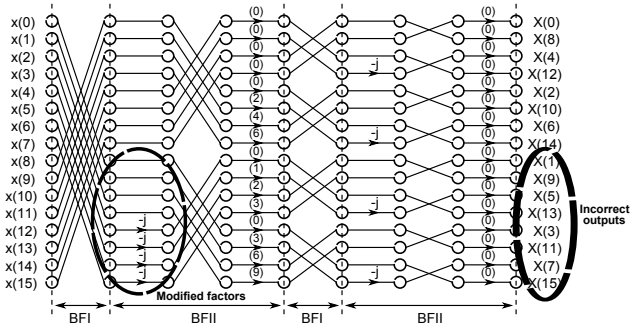


Figure 2.6: Effect of modification of -j multiplication factors on output

tem is working in the 2-parallel mode, the outputs are obtained in the order $X(0)/X(8)$, $X(2)/X(10)$, $X(1)/X(9)$, $X(3)/X(11)$, $X(4)/X(12)$, $X(6)$ $/X(14)$, $X(5)/X(13)$, $X(7)/X(15)$. Now if we modify the first column of $-j$ multiplications such that only the top 50% are correct values, we see that the outputs $X(0)/X(8)$, $X(4)/(12)$, $X(2)/X(10)$ and $X(6)/X(14)$ are correct and the rest are incorrect. Such a modification corresponds directly to changing the control signal t of the first BFII block of the obfuscated FFT architecture. The details of change in operations that occur for a single block are shown

in Figure 2.7.

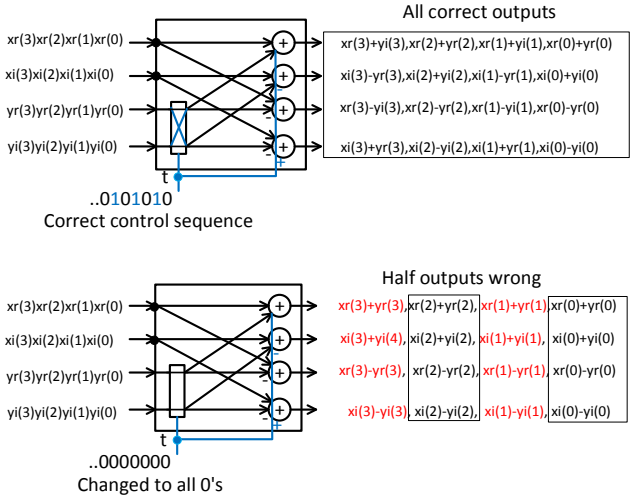


Figure 2.7: Modification of control logic of BFII (input t)

Thus, we have created a partial mode with 50% correct outputs. Various such combinations can be used to generate non-meaningful modes which correctly compute 25% and 50% of the outputs.

2.3 Complete design

We now discuss usage of concept of folding and control-flow modifications to create a complete obfuscated design. Since these approaches can be applied in various ways, we can create a reconfigurable and flexible design which can be tailored according to the level of security desired and the acceptable overhead.

2.3.1 Design of meaningful modes

We make use of the concept of folding on radix-2^2 FFT flowgraphs to generate two-parallel FFT architectures of varying lengths. For example, we could design a system which can implement four architectures such as 16-point/64-point/256-point and 1024-point FFTs. Various folding sets can be used for this purpose. One of the four architectures obtained as a result of folding is shown in Figure 2.8.

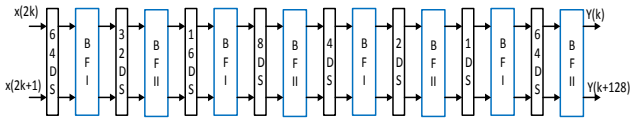


Figure 2.8: 256-point, 2-parallel folded FFT architecture

We now proceed to integrate the four architectures to one structure which can be programmed to operate in four different modes. We choose architectures such that most of the blocks of individual architectures overlap with the merged architecture. The only modifications necessary are in terms of additional delay-switch-delay elements and muxes. The 16/64/256/1024-point obfuscated FFT after these changes is shown in Figure 2.9. Corresponding select signals which are derived from the key can then be used to select one among the four modes using the inputs $c0 - c3$. Thus, we have

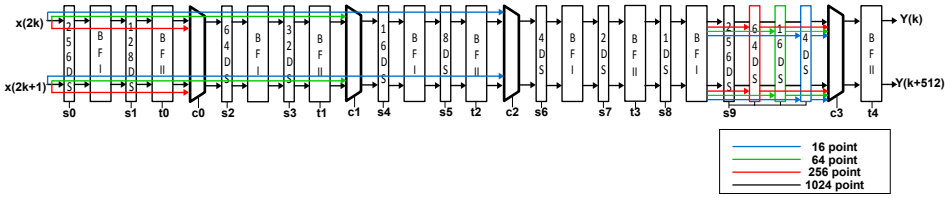


Figure 2.9: Obfuscated FFT with four meaningful modes built on 1024-point, 2-parallel folded FFT

generated a structure which operates in four different functionally correct modes which we call as the *meaningful modes* of the design. It is to be noted, that we can design the obfuscated FFT to work anywhere between 1 to 4 meaningful modes with associated trade-offs as will be discussed later.

2.3.2 Design of non-meaningful modes

Next, we use the concepts of control-flow modifications on switches and trivial multiplication selection of butterfly units as discussed earlier. From Figure 2.9, we observe that the control signals available for modification include $s0 - s9$ of switches and $t0 - t4$ of butterfly units. The correct sequences for each of these signals are derived from a 10-bit counter and depend on the location and associated delays of the architecture as described in [21]. However, as part of obfuscation, additional sequences are also derived

from the same counter. Since, we have seen that changes in the sequence for these signals affect the output in different ways, the derived incorrect sequences can be used to obfuscate the design. This concept is illustrated in Figure 2.10.

In Figure 2.10, cntr represents a 10-bit register and cntr[i] represents its i-th bit. Note that $s0$ has four different sequence combinations but only one value, i.e., cntr[8] is the correct sequence for $s0$ when operated in 1024-point FFT mode. The other sequence combinations are derived to generate either random sequences or bits 0 and 1, all of which affect outputs. In this example, all other signals are also obfuscated with 4 different sequence choices. However, it is to be noted that the number of sequence derivations for each of these signals can be increased and this results in an increase in the size of the mux at the output of the signal. Increase in the size of the mux from 4:1 upto 32:1 results in a flexible design method with associated trade-offs as will be discussed later.

Once these obfuscated signals are set up, the select input for each of the muxes can be derived from the key of the design. A correct key selects all correct control signal combinations. Rest of the keys are mapped to modes which are non-meaningful or partially correct by using various other control signal combinations. Hence, depending on the size of the key, non-meaningful modes are incorporated into the design. For each of these non-meaningful modes, it is advised to have atleast 50% deviation from the correct control signal combination.

For example, we can create a non-meaningful mode which selects the correct control signals for $s[0], s[2], s[4], s[6], s[8]$, $t[0], t[2]$ and incorrect signals for the rest. This produces a mode with all incorrect outputs. On the other hand, if we select correct control for all signals except $t[0]$ and $t[4]$, we can create a partially correct non-meaningful mode with 25% correct outputs when made to operate in a 256-point, 2-parallel mode. Thus, control information known only to the designer is exploited in designing obfuscated modes.

2.4 Analysis of obfuscated modes

The effectiveness of this obfuscation scheme is analyzed by defining an attack model and then considering the role played by modes in obfuscation. It is important to note that

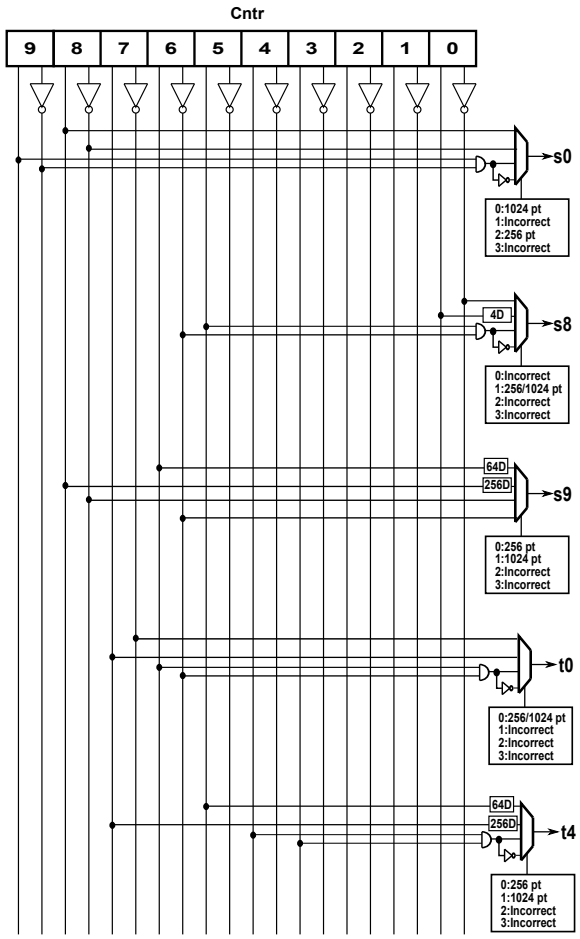


Figure 2.10: Obfuscated control-path with various signals derived from a counter

several types of attack models exist and the attacker could be using more sophisticated techniques and tools. However, in this work, we make use of an attack model generally used in design of obfuscation schemes and which reflects the scenario of an IP piracy or reverse engineering based attack [22].

2.4.1 Assumptions of the attack model

The design flow as described earlier is used to create an obfuscated FFT incorporating several different meaningful and non-meaningful modes. Since the FFT is used as a submodule in several applications, we assume that a complete architecture for the specific application is built around the obfuscated FFT. A netlist is then generated and the design is sent for further manufacturing to untrusted foundries. Once the design is manufactured and sent back to the design house, the key is programmed and stored in a safe memory inside the chip. This key selects one of the meaningful modes of operation of the circuit.

We assume that the attacker could get access to the obfuscated netlist through various sources and hence the circuit could be subjected to structural observations and functional simulations. The goal of an attack would be to try to decipher the correct key and in turn get access to the correct control information of the system. A second goal would be to guess what length of the FFT is used in the final design of the integrated circuit. Functionally correct I/O pairs can be assumed to be available to the attacker from a functional IC. However, this is from the application in which the FFT is used and hence does not give any additional information.

2.4.2 Obscurity of control-flow

Let us assume there are C different control signals necessary for the design. Each of the signals is obfuscated to a degree L using techniques described previously. For this purpose, a L:1 mux is used at the output of the control signal. For example, from Figure 2.10, $L = 4$ and from Figure 2.9, $C = 15$ corresponds to $s = 10$ and $t = 5$ variables. The strength of this method stems from the observation that it is not possible to derive the correct select signal for this $L : 1$ bit mux unless a complete simulation of the system is performed. Thus, we have L^C different control signal combinations possible out of

which only one is correct. For a circuit with a large number of control signals, a smaller value of L can be used to achieve the same level of obfuscation,

At the next level, we consider the selection of these signals to create modes. For each mode, we can use M incorrect signal selections. A value of M equal to 0 would mean a meaningful mode of operation. For each control signal change at the switches or butterfly units, the outputs are deviated from actual outputs by either 100% or 25-50% , respectively. Even though, individual control signal changes cause sufficient difference in Hamming distance between correct and incorrect outputs, we use a value of $M \geq C/2$ to create non-meaningful modes. This ensures at least 50% deviation from the correct control sequence combination making it secure against attacks. Thus, for each M value, we can modify the signal incorrectly in L ways using the mux. This gives us a total of $\binom{C}{M} * L^M$ modes, for every chosen value of M. The number of modes in turn dictates the size of the key and can be increased by using various values for M. Thus, control information is hidden effectively through the correct choice of L and M.

2.4.3 Protection of length of FFT

Meaningful modes and partially correct modes help in protecting the length of the FFT. Upon observation of an obfuscated netlist, an attacker is able to see the obfuscated datapath consisting of hardware units for four meaningful modes of the design. However, the attacker has no way of understanding which among the four modes has been used in the application. Thus, these modes successfully defend a reverse engineering attack. The partially correct modes give out some information about the system like 25% to 50% correct outputs. This can also be used to mislead the attacker into believing an incorrect length of the FFT is being used. For example, if 25% of the outputs of a 256-point FFT are observed by an attacker in a particular mode, then one might be led to believe that the application uses a 256-point FFT when actually a 1024-point FFT is used. Additionally, the partial control information given out in this mode is that of the incorrect length, i.e., 256-point FFT.

Table 2.1: Overhead due to meaningful modes

No. of meaningful modes	**Total area overhead**	**Total power overhead**
1 (1024 point)	0.2%	0.5%
2 (256/1024 point)	8.5%	10.6%
3 (64/256/1024 point)	38%	15.5%
4 (16/64/256/1024 point)	41.5%	17.3%

2.5 Area and power overhead

We quantify cost associated with the obfuscation scheme by simulating and synthesizing architectures with various modes and key sizes and calculating area and power overheads. All architecture descriptions are written using Verilog HDL and synthesized using Design Compiler. A $65nM$ technology library is used for all synthesis and the circuits are clocked at a speed of $100MHz$. The overheads are compared with a 1024-point folded FFT design but with no obfuscation.

By varying several parameters of the design, we present three separate sets of results. For the first set of analysis, the datapath is obfuscated to have several meaningful modes ranging from 1 to 4, while keeping the switch size of controlpath at a value 4 and key size as 16. It is to be observed that the meaningful modes involve changes in the datapath of the design and hence incur larger overheads with each additional mode. This is tabulated in Table 2.1. Hence, for practical applications it would be advisable to keep the number of modes at a nominal value of 2 for this FFT architecture. Security critical applications could use higher number of meaningful modes.

Next, we vary the switch at the control path outputs added as part of obfuscation. It was seen that these switches mainly contribute to the obscurity of control-flow of the design and hence its security. From Table 2.2 it is seen that the size of muxes ranging from 2 to 16 cause an overall overhead increase of around 2%. Hence, it can be used

Table 2.2: Overhead due to size of mux at controlpath

Mux size of controlpath (L)	**Controlpath overhead**		**Total overhead**	
	Area	**Power**	**Area**	**Power**
2	2%	0.7%	8.2%	10.1%
4	5.5%	1.7%	8.35%	10.5%
8	22%	4.2%	8.5%	11.4%
16	41%	6.7%	9.1%	12.1%

Table 2.3: Overhead for different key sizes

Key size (Modes)	**Total Overhead**	
	Area	**Power**
4 (16)	8.29%	10.53%
8 (256)	8.33%	10.55%
16 (65536)	8.35%	10.59%
20 (1048576)	8.42%	10.63%
28 (268435456)	8.47%	10.66%

as a parameter to adjust levels of obfuscation as necessary. For these simulations, a meaningful mode value of 2 and key size 16 was used.

Finally, for the last set of results we keep the meaningful modes as 2, a switch size of 4 and vary the length of the key. For each of the keys, modes are created by using signal combinations from the obfuscated design. The number of incorrect signals is kept at a nominal value of 50%. It is seen that the area and power overheads do not vary much for key sizes upto 28 and hence it is easy to build the architecture to accommodate large key sizes.

Comparison of overheads with general encryption or obfuscation schemes cannot be made, since these schemes do not specifically address the goals of DSP circuit obfuscation. This includes protection of important properties such as length of FFT. However, still with a nominal meaningful mode value of 2, an area overhead of 8% and

power overhead of 10%, comparable to general obfuscation or encryption schemes is obtained [17, 22]. An automated method could be developed which takes as input the key size, switch size and number of meaningful modes and tries to meet a specified area and power constraint.

2.6 Conclusion

In this work, we have successfully demonstrated the mode-based method of obfuscation of circuits using a complex FFT architecture. We also illustrated a new approach to design modes, specifically using control-flow changes to design non-meaningful modes. The role of various modes with respect to security of the circuit and obfuscation achieved was also analyzed. Finally we showed that despite the method serving as a strong obfuscated FFT architecture, it does not increase the overhead of the design significantly which makes it practical for use in current designs. Future work would involve developing metrics to measure the level of obfuscation of a design. Such a metric would not only be a measurement tool but would also serve as a comparison tool for several approaches to obfuscation existing today.

Chapter 3

Dynamic obfuscation

This chapter presents a novel technique for hardware obfuscation termed *dynamic functional obfuscation*. Hardware obfuscation refers to a set of countermeasures used against IC counterfeiting and illegal overproduction. Traditionally, obfuscation encrypts semiconductor circuits using *key* inputs which must be set to a correct value to operate the circuit correctly. By keeping the key values secret during the manufacturing process, any attempt by unauthorized parties to overproduce chips or pirate designs is thwarted. The proposed dynamic technique differs from existing fixed obfuscation schemes as the obfuscating signals change over time. This results in inconsistent circuit behavior upon input of incorrect key, where the chip operates correctly sometimes and fails sometimes. The advantage of dynamic obfuscation is that it results in stronger obfuscation by increasing the time complexity of deciphering the correct key using brute-force attack, even with shorter keys. Moreover, the dynamic nature of these circuits also makes them resistant to reverse engineering and SAT solver based attacks. To achieve dynamic obfuscation, ideas from hardware trojan literature and sequentially triggered counters are utilized. A demonstration of obfuscation on sequential circuits implementing fast Fourier transform (FFT) algorithm and Ethernet IP shows low overall area and power overheads of less than 1%. Security in terms of time to attack for the FFT circuit (for a key size of 30 bits and a system operating at 100 MHz) is increased to 1,021,055 years using dynamic obfuscation compared to only 5.36 s using fixed obfuscation schemes. For the Ethernet IP core, time to attack of dynamic obfuscation with a key size of 32 bits is 1,046,423,135 years compared to 21.47s with fixed obfuscation. It is also shown

that for a key size of K bits, the lower bound for time to attack using brute-force is proportional to $K2^K$ and $K2^{2K}$ for the proposed design using one and two random number generators, respectively.

3.1 Introduction

Ensuring security of devices during design and manufacturing is of paramount importance in modern product life cycle management. With production of integrated chips involving various companies located in different countries, this can quickly become a daunting task. Issues such as counterfeiting, unauthorized overproduction, piracy etc. have created a huge loss of revenue for semiconductor manufacturing companies [23,24]. Also, threats such as reverse engineering and malicious circuit insertion have added to the burden of ensuring reliable and safe circuits.

Adding encryption to a system by making use of secret keys and trying to keep its design and operation protected contribute to a countermeasure termed obfuscation. *Hardware obfuscation* can be defined as hiding the functionality of a design and preventing black box usage by making use of secret keys. In cryptography, encryption involves converting a plaintext into ciphertext using a secret key and passing it through untrusted channels. At the receiver end, the message is decrypted using the same secret key. Similarly, the circuits are obfuscated during design, where the key is only known to the design house. The foundry does not have access to the key. After fabrication, the keys are programmed either by the design house or a trusted party appointed by the design house. This analogy is represented in Fig. 3.1.

This work addresses functional obfuscation where the functionality of the circuit is controlled by keys. Functional obfuscation is based on the observation that circuits that look alike may compute different functions and circuits that look different may compute the same function [25]. Functional obfuscation is different from logic encryption where keys are used as inputs to logic gates to obfuscate the random logic [16]. Though there are different techniques to hardware obfuscation in current literature [16,17,26–29], these suffer from several shortcomings. Attacks on circuits embedded with these obfuscation techniques have proven that the current methods are not fully secure. While some methods are difficult to design and implement, others cause overheads in terms of area,

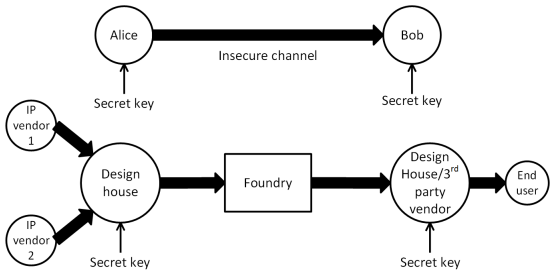


Figure 3.1: Comparison of encryption scheme as defined in cryptography with hardware obfuscation technique for protection during manufacturing

power or by increasing timing requirements.

In this regard, we propose a new technique to obfuscation termed *dynamic obfuscation*. In dynamic obfuscation, unlike in existing techniques, the circuit behaves incorrectly *sometimes*. Note that in *fixed* obfuscation, an incorrect key results in incorrect outputs for *all* time instants. Dynamic obfuscation has several advantages including stronger security and smaller key size requirements. The dynamic obfuscation technique is inspired by research on hardware trojans. As part of security analysis, two attacks based on *brute-force* and *reverse engineering* are discussed in detail. The techniques discussed are applied to sequential circuits and the corresponding trade-offs with respect to area and power overheads are presented. Comparisons between the dynamic and existing obfuscation schemes are also discussed.

3.2 Related work

Several hardware obfuscation techniques have been developed over the last few years targeting both combinational and sequential circuits. While some publications address these schemes as obfuscation, others call it encryption since the concept has evolved into using a key to lock the design rather than the traditional definition of hiding functionality. Recently, a new terminology of logic locking has also been used to describe these schemes. Regardless of the terminology, the basic idea of obfuscation is to secure the design using keys and prevent unauthorized usage.

Some of the early papers on logic obfuscation have been published in [26] and [17]. Both these techniques are based on insertion of finite state machines referred

to as *obfuscating FSM*. The obfuscating FSM accepts key bits as input and selects the states of the FSM. Only upon activation of the correct states, the circuit or its components are able to function correctly. These obfuscation techniques assume that the obfuscated netlist of the design is not available to an attacker. However, practical reverse engineering attacks [30] have invalidated these assumptions. It means that the obfuscating FSM can be traced and just removed from the design and then the circuit can be copied or misused.

Combinational logic encryption by inserting key gates in the design was introduced in [16]. The key gates correspond to additional gates such as xor/xnor added for obfuscation and mapped to key bits. Correct key passes the correct signals in the design giving a correct output. For incorrect keys, the signals get modified and incorrect outputs are produced. Choosing the correct location of these gates to avoid fault sensitization based attacks was discussed in [27]. Instead of using xor/xnor gates, multiplexers were used in [28] with the key bits mapped to the select signal of the mux. Correct and incorrect signals are input to the mux and chosen based on the key. Another technique to insert key gates at nodes such that controllability is minimized using and/or gates was presented in [29]. A recent work on attacking logic encryption using SAT solver based tools [31] exposed the shortcomings of logic based encryption techniques using xor/xnor, and/or and mulitplexers. This has led to several works to thwart this attack including [32] but has its own shortcomings with respect to having high overheads.

The works described in [33–35] combine FSM of obfuscation with Physically Unclonable Functions (PUF) to create circuits whose states are dependent on the PUF output. Since there is a dependence on PUF output, each IC has its own unique signature and this scheme is also referred to as IC metering. After manufacturing of chips, each of the individual ICs is tested to obtain information necessary for unlocking the chip and this information is sent back to the design house. Combining this information with the knowledge of the design, only the design house can then unlock each chip and keep track of the number of authentic chips released in the market. On the other hand, obfuscation adds a secret key to each batch of chips being manufactured. The key is then programmed onto the chip by the design house and safely stored in a flash memory location. We do not address IC metering in this work but believe that both these techniques have advantages and disdvantages. While the key stored on the chip might

be prone to fault attacks in obfuscation, the data required for identifying and tracking each IC might be a challenge with respect to processing and transfer in metering. Apart from IC metering, gate-level obfuscations have also been proposed that involve making changes to the basic structure of the gates [36].

With respect to sequential logic obfuscation, an algorithm applicable to digital signal processing circuits such as filters and FFT, using a high level transformation approach was proposed in [37]. This technique made use of meaningful and non-meaningful modes of the design to hide its functionality and make it unusable. Specific applications of this technique using FFT circuits were discussed in [3,19]. However, these methods are very specific to the circuits discussed and have not been demonstrated in general. In this work we use concepts from these schemes and present a new methodology for obfuscation. In the proposed *dynamic obfuscation* scheme, the datapath of the system remains intact. The modes are controlled by changing the control circuits only.

A technique to protect against side-channel attacks on scan cells which are included in a chip to aid testing, has been proposed in [38]. The obfuscation applied to scan data has been termed dynamic obfuscation. However, the technique of dynamic obfuscation proposed in [20] is different from what is proposed in this work. Also, a technique for protection of IPs released for evaluation purposes using hardware trojans has been presented in [39]. Even though our work also uses the same high level idea of dynamically modifying data for protection against attacks and hardware trojan circuits for the design of dynamically obfuscated circuits, the application of these concepts to traditional hardware obfuscation which protects the chip during manufacturing has not been addressed in any prior publication.

3.3 Fixed, Time-Varying and Dynamic key-based obfuscation

3.3.1 Basic Concepts of Key-Based Obfuscation

In a multiplexer based hardware obfuscation, 2-input multiplexers are used to insert key bits into the design [28]. The key bits are mapped to the select signals and correct

and incorrect signals are mapped to inputs of the multiplexer. The outputs of the multiplexers in the proposed approach represent control signals for the circuit. Therefore, the two inputs to the multiplexer, respectively, correspond to the correct and obfuscated control signals. For example in Fig. 3.2, the correct signals are $C1$ and $C2$ and the incorrect signals are $C1'$ and $C2'$ mapped to two key gates with select bits $K[0]$ and $K[1]$. When $K[0] = 0$ and $K[1] = 1$, the correct signal combination (marked in blue) is obtained at $S1$ and $S2$. For all other key bits, incorrect signals are selected. We term this method of obfuscation as *fixed obfuscation* in this work.

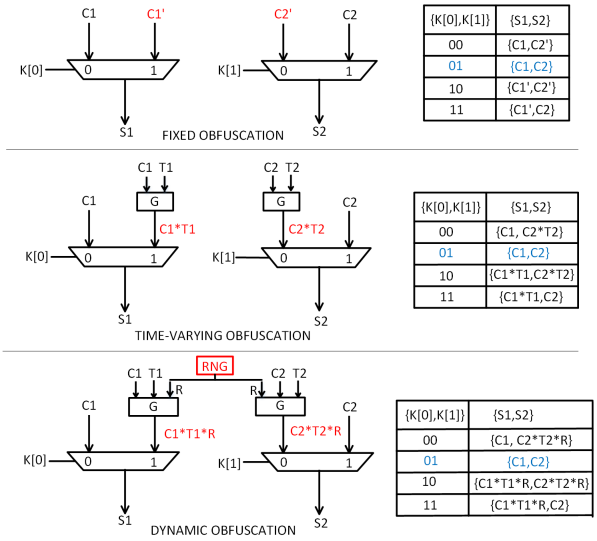


Figure 3.2: Difference between fixed, time-varying and dynamic obfuscation using multiplexers (Correct key values are $K[0] = 0$ and $K[1] = 1$)

In the proposed *time-varying obfuscation* approach, the incorrect key values for $K[0]$ and $K[1]$ select the signals $C1 * T1$ and $C2 * T2$ at $S1$ and $S2$, respectively. $T1$ and $T2$ are termed *trigger* signals. The notation $Ci * Ti$ represents an obfuscated signal that is derived using the correct control signal Ci and a trigger signal Ti , for i =1 and 2. The function G represents this combination. In the proposed *dynamic obfuscation*, a random number is combined with the trigger signal to break its predictability. The notation $Ci*Ti*R$ represents a signal that is a function of the control signal Ci, trigger signal Ti and the random number R. We discuss approaches for fixed, time-varying and dynamic obfuscation before we delve into the details of generating triggers.

3.3.2 Mode based Control-flow modification

In this section, we introduce the technique of fixed obfuscation of sequential circuits based on *mode-based obfuscation*. The concept of mode-based obfuscation was first introduced in [3] for a specific example of folded, Radix-2^2 FFT algorithm. We reiterate this method briefly and discuss how to transform the design into a dynamic obfuscation based design. Typically, architectures are partitioned into data-path and control-path to help facilitate the testing and optimization of designs. The control-flow of the design decides its correct operation and this information is most critical to such systems. Hence, we try to use this critical link in the design to introduce multiplexers controlled by key bits for obfuscation. Since the introduced key bits make the control-flow obscure to an attacker without access to the correct key, this design technique is referred to as *mode-based* control flow modification [3].

A top level overview of control flow modification is illustrated in Fig. 3.3. The signals generated by the controlpath and received by the datapath are derived from counters. Along with correct control sequences, incorrect control sequences are also derived using counter bits and the inverted counter bits. These together serve as the inputs to the multiplexers added for obfuscation. The select signals of the multiplexers are mapped to keys. In this design, the number of key bits that can be mapped depends on the number of control signals available for obfuscation and the size of multiplexers. For example, if there are 10 control signals available and if each of them is obfuscated using a 4-input mux (using a 2-bit select signal), then 20 key bits can be mapped to the control signals.

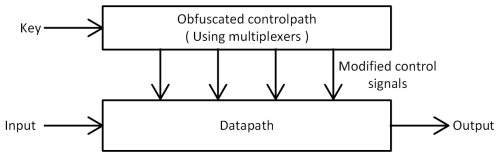


Figure 3.3: High level overview of mode-based obfuscation using control-flow modifications using multiplexers

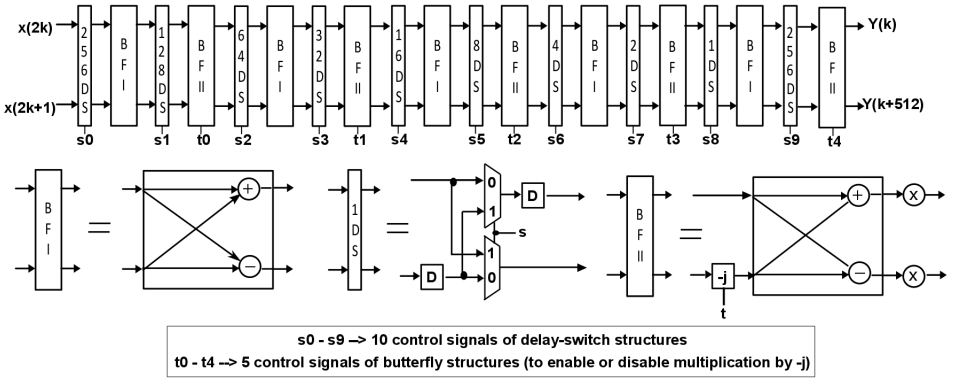


Figure 3.4: Datapath, components and required control signals of a Folded, Radix-2^2 1024-point FFT

3.3.3 Example of mode-based obfuscation

To understand control-flow obfuscation better, we consider an example of a sequential circuit used to implement the fast Fourier transform (FFT) algorithm. Two other examples will be discussed in Section VI, when we present results of the proposed techniques. The FFT architecture used for our study is a folded, radix-2^2 Decimation-in-Frequency (DIF) architecture [21]. The 1024-point FFT architecture is depicted in Fig. 3.4 and shows the basic components of the design namely Butterfly units (BFI and BFII) and Delay-Switch units (DS). The control signals for the correct operation of this circuit include $s0 - s9$ of the switches and $t0 - t4$ of butterfly structures as depicted in the figure. A 10-bit counter is used to derive the required control signals of the design. The 15 signals of the controlpath can be altered by inserting multiplexers as part of obfuscation to create modes.

Modifications to the control signals described previously create modifications in the final output of the design. Specifically, the effect of changes to control bits of switches $s0 - s9$ is shown in Fig. 3.5. For example, a change of the signal to all 1's or all 0's obfuscates the outputs of the delay-switch circuits. Similarly, the signals $t0 - t4$ of butterflies can be modified to produce corrupt outputs as shown in Fig. 3.6. These observations suggest that each of these incorrect signals can serve as inputs to the key-gate multiplexers. For the 1024-point FFT, the control signals and their modifications

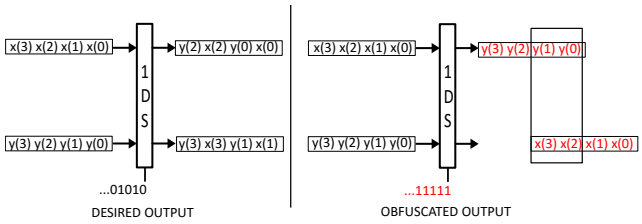


Figure 3.5: Incorrect outputs due to modification of control s of switches

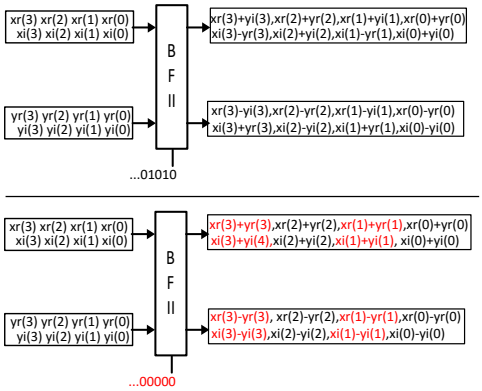


Figure 3.6: Corruption of outputs due to modification of control t of butterfly units

are derived from a 10-bit counter. An example of a control signal derived from the counter and its various modifications mapped to a key-gate are illustrated in Fig. 3.7. Since the correct key values to all multiplexers in the design result in correct operation, the system is said to operate in a *meaningful mode* for the correct key. For all other key values, computationally incorrect outputs are generated, resulting in the system operating in *non-meaningful modes*.

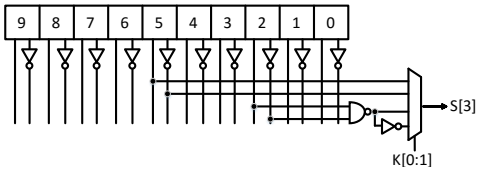


Figure 3.7: Deriving correct and modified control signals from counter and mapping of key bits

3.3.4 Time-varying and Dynamic sequentially-triggered modes

To convert the fixed mode obfuscation into a dynamic one, we introduce trigger circuits into the design. It is to be noted that the trigger circuits generate signals which trigger on rare conditions and in a periodic manner. Integrating the mode-based design with the trigger circuits is illustrated in Fig. 3.8. Instead of the modifications to control signals derived previously, here the control signals are obfuscated with the trigger signals using a *trigger combination* circuit. The obfuscated and correct signals are then input to multiplexers. A multiplexer connected to one of the control signals is shown in the figure as an example.

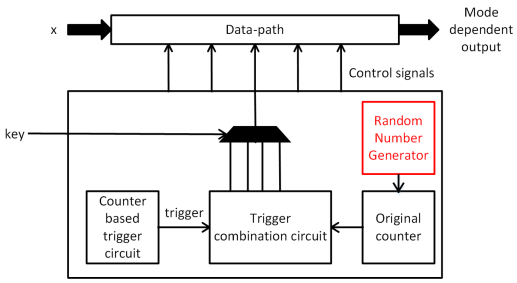


Figure 3.8: Trigger circuit combined with mode-based obfuscation to create time-varying and dynamic obfuscation

The non-meaningful modes of the system now behave differently due to the presence of trigger circuits. When the trigger is on, outputs computed are incorrect. For the rest of the time, the system computes computationally correct outputs just like a meaningful mode. Hence, with each key value input to the system, the system behaves in a time-varying manner making the modes *time-varying*. Any attack involving the traversing of key space to decipher correct key values is made difficult by the fact that the behavior of the system is not constant. To increase the complexity of the system to an attack, randomness is introduced into the modes. The output of a random number generator is input to the trigger generation circuit to break the periodicity of the trigger signal. For every incorrect key value, instead of triggering at a fixed time duration, the trigger circuit is made to trigger at random. Thus, the modes become *dynamic* such that even if the same key value is input to the system, it behaves differently.

For each of the incorrect dynamic modes, it is required that the trigger occurs infrequently. Thus, the *trigger period* of the trigger signal needs to be large. Also, significant damage should occur when the trigger is on, making the circuit unusable. This requires that the *duty cycle* of the trigger signal be sufficiently high. Furthermore, it is necessary that not too many of the outputs are incorrect which would make it easy to identify the incorrect mode. These requirements are kept in mind while designing the trigger circuits. A second important consideration is the combination of the trigger circuits with the original design. Modern reverse engineering attacks assume that structural inspection of circuits is possible. Thus, the trigger circuits need to be incorporated such that they evade detection and are not traceable from the primary inputs or outputs. This is made possible by the careful design of *trigger combination* circuits. These two important ideas are discussed next.

3.4 Trigger circuits and trigger combination circuits

3.4.1 Design of trigger circuits

Inspired by the concept of sequentially-triggered, hybrid *hardware trojans*, we design trigger circuits which activate on rare conditions and are difficult to detect. We start with a simple design considering a key size of 2 bits. The subsections then demonstrate how to extend the design for larger key sizes.

Counter based hardware trojans

Hardware trojans are an emerging threat to modern day integrated circuits. Trojans constitute any malicious modification to hardware leading to altered functional behaviour [40]. These circuits are difficult to detect since they are small, have a stealthy behavior and get activated on conditions which are difficult to recreate while testing. Upon activation, trojans cause damage to the circuit by corrupting outputs or physically damaging its structure. The insertion of trojans leads to a large variety of structures and operating modes in the design. Also, unless the trojan circuits create a large difference in the power and timing of the original design, it is easy for them to escape detection.

There has been substantial research on the detection of hardware trojans and countermeasures that can protect against them. Efforts have been taken by various groups

to understand and classify the different types of trojan circuits. In this work, we look at the design of a sequential hardware trojan and utilize similar concepts for our design. Sequential trojan circuits can be synchronous, asynchronous or hybrid as discussed in [41]. Simplified circuit diagrams of these sequential trojans are shown in Fig 3.9. In

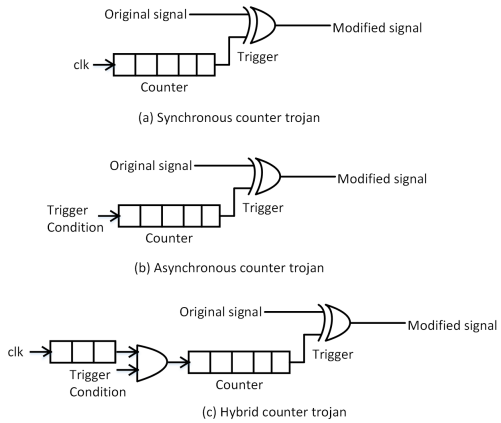


Figure 3.9: Different types of hardware trojan circuits based on the method of obtaining trigger signals

a synchronous counter trojan, the trigger occurs after the counter reaches a pre-defined value. This is time based and hence termed "time-bomb" trojans. In an asynchronous counter trojan, instead of incrementing the counter for each clock cycle, the counter increments based on internal trigger conditions. Finally, the concept of synchronous and asynchronous trojans can be combined to produce a hybrid circuit. This design uses both clock based counters and asynchronous trigger condition to determine when the trojan activates. Once the activation occurs, the trigger modifies the original signal to behave differently from its original intent.

Even though hardware trojans are malicious and dangerous for designs, a defender could use them for effective design obfuscation. As the defender is aware of the complete architecture of the circuit, in comparison to the attacker, it is easier for him/her to add trojan-like circuits in places difficult to detect. The circuits can then be used to control obfuscation of the design using keys. Trojan circuits that have been identified in literature are used to our advantage here.

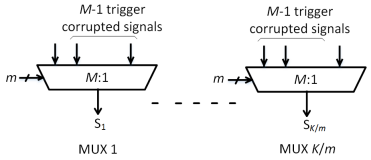


Figure 3.10: Number of multiplexers and Trigger signals required to map a key size of K bits

Trigger circuits using counters

Before considering the design of trigger circuits, we list some assumptions based on the desired properties of trigger signals and mapping of key bits as illustrated in Fig. 3.10.
Assumptions:

1. To map a key of size K using C control signals which are available for obfuscation, select signals of size $m = \lceil \frac{K}{C} \rceil$ and multiplexers of size $M = 2^m$ are used. For simplicity, we assume K is a multiple of C.

2. At most one trigger signal is active at any instant of time.

3. For each M : 1 multiplexer, the number of trigger signals required is equal to $M - 1$.

4. To map K key bits, $\frac{K}{m}$ M : 1 multiplexers are used and the number of trigger signals required is equal to $\frac{K}{m}(M - 1)$.

5. A trigger signal can be selected by multiple key values. The number of times a trigger gets selected at each multiplexer is $\frac{1}{M}$ and the total number of key values are 2^K. Hence, for each trigger signal this value is given by $\frac{2^K}{M}$.

For example, for a key of size 2 bits ($K = 2$), a multiplexer based obfuscation would require two 2-input muxes ($M = 2, m = 1$) and trigger signals $T1$ and $T2$ ($\frac{K}{m}(M-1) = 2$) as was illustrated in Fig. 3.2. T1 is selected by key values 10 and 11 and T2 is selected by key values 00 and 10 ($\frac{2^K}{M} = 2$). The design of a sequential trigger circuit to generate these trigger signals using counters is shown in Fig. 3.11. A part of the existing circuit is used for *activation*. The activation part could be purely combinational or a counter activating periodically. Let us assume that the activation circuit is activated with a

period L. For example, L can be the number of clock cycles required for completing an N-Point FFT. Other values of L can be used in general.

The activation then triggers a counter, termed *delay counter*. The delay counter has an output **cntd** and controls the delay after which a trigger would activate a final counter. The *final counter* increments every time the delay counter overflows. The output of this counter, **cntf**, and cntd are input to a *trigger generator* circuit. A third input **rng** is also provided to the trigger generator from a random number generator. The trigger generator provides signals $T1$ and $T2$.

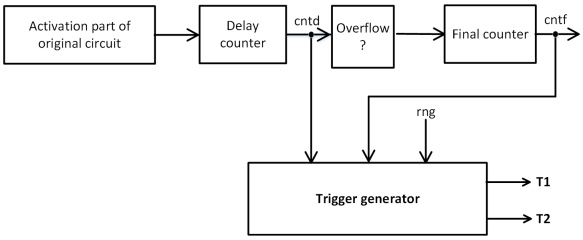


Figure 3.11: Structure of a sequential trigger generation circuit which uses a series of counters (cntd, cntf) and a random number generator output (rng) to generate triggers $T1$ and $T2$

The generated signals $T1$ and $T2$ depend on the trigger conditions of the trigger generator. For example, consider the following conditions:

If (cntf == rng and cntd==0) $\rightarrow$ $T1$=1, else $T1$=0

If (cntf == rng and cntd==1) $\rightarrow$ $T2$=1, else $T2$=0

Design parameters:

- **Size of delay counter (*size_cntd*)**: We observe that to ensure Assumption 2 holds, each of the trigger conditions should be unique. Thus, the number of states of the delay counter should be equal to the number of trigger signals required. Using Assumption 4, this value is given by Equation (3.1).

$$\begin{aligned} size\_cntd &= log_2(\frac{K}{m}(M-1)) \\ &= log_2(K) - log_2(m) + log_2(M-1) \\ &\approx log_2(K) - log_2(m) + m \ for \ m > 1 \end{aligned} \tag{3.1}$$

- **Size of final counter ($size\_cntf$)**: The final counter is compared to the random number generator in each trigger condition. Hence, the value of size of final counter is given by Equation (3.2).

$$size\_cntf = size\_rng \tag{3.2}$$

- **Size of random number generator ($size\_rng$)**: Since multiple key values can select the same trigger signal, the trigger occurrence of each trigger signal is randomized. To minimize the probability of detecting multiple key values using the same trigger signal, we choose the number of states of the random number generator to be greater than or equal to the number of key values selecting a trigger signal. From Assumption 5, this value is given by $\frac{2^K}{M}$. Hence, the size of the random number generator is given by Equation (3.3).

$$\begin{aligned} size\_rng &= log_2(\frac{2^K}{M}) \\ &= log_2(2^K) - log_2(2^m) \\ &= K - m \\ &\approx K \end{aligned} \tag{3.3}$$

For the example of key size of 2 bits, we select the size of delay counter to be 1 bit and the sizes of final counter and random number generator to be 2 bits. Using these values, trigger times for $T1$ and $T2$ for different values of rng and all possible key combinations are shown in Fig. 3.12. In this figure it is assumed that the random number generator is held constant for a time equal to the trigger period of the signal (which will be discussed later) and has values from 0 to $2^{size\_rng} - 1$. Hence, a trigger occurs every 8 cycles. In practice, an LFSR which is changing every clock cycle and does not reach the value 0 is used to generate these random numbers. Also, the value of L is equal to 1 cycle for simplicity. Even though multiplexers of any size can be used, for most practical purposes a 2:1 or 4:1 mux will suffice. Hence, we consider only these two cases in this work.

Trigger period, Duty cycle and Percentage of incorrect outputs

The design of trigger circuits using sequential logic results in periodic signal outputs. The trigger period of these signals determines how often the corruption in output occurs.

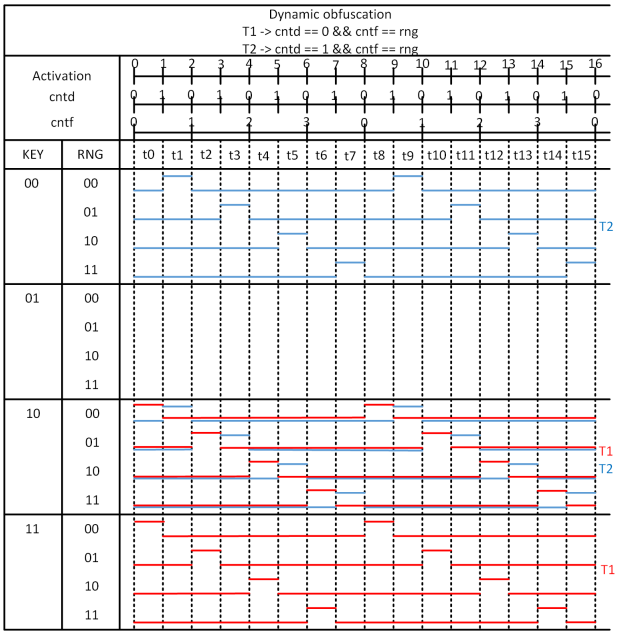


Figure 3.12: Timing diagram of trigger signals $T1$ and $T2$ based on conditions used in the trigger generation circuit

To avoid detection, this period should be as long as possible. The calculation of trigger period is given by Equation (3.4).

$$\begin{aligned} trig\_period &= L \times 2^{size\_cntd} \times 2^{size\_cntf} \\ &= L2^{log_2 K} 2^K = LK2^K \ (using\ 2:1\ mux) \\ &= L2^{log_2 K+1} 2^K = LK2^{K+1} \ (using\ 4:1\ mux) \end{aligned} \tag{3.4}$$

The duty cycle of trigger period determines how long the trigger is active. The duty cycle should be sufficient to create enough damage to the circuit to make it inoperable. The duty cycle is calculated as the ratio of the on period of the trigger signal to its total period. Since the trigger is dependent on cntd, the trigger signal is on for one full increment of the delay counter, which happens every L cycles. The duty cycle is thus

given by Equation (3.5).

$$\begin{aligned} duty\_cycle &= \frac{L}{trig\_period} \\ &= \frac{1}{K2^K}\ (using\ 2:1\ mux) \\ &= \frac{1}{K2^{K+1}}\ (using\ 4:1\ mux) \end{aligned} \tag{3.5}$$

Finally, the average percentage of incorrect outputs for each key is given by Equation (3.6). A detailed derivation of the proof of this equation is provided in Appendix A.

$$\begin{aligned} avg\_percent &= \frac{100}{2^{K+1}}\ (using\ 2:1\ mux) \\ &= \frac{100}{2K}\sum_{n=1}^{\frac{K}{2}} n3^n \binom{\frac{K}{2}}{n}\ (using\ 4:1\ mux) \end{aligned} \tag{3.6}$$

Configurations for all key sizes starting from 2 bits upto 32 bits are tabulated in Table 3.1. We see that the trigger period quickly reaches high values with the increase in key size. The trigger period has a direct dependence on the value of L, which can be

Table 3.1: Trigger period, duty cycle and percentage incorrect outputs for different key sizes and L=100

Key size	**Mux size**	**Counter size {size_cntd, size_cntf}**	**Trigger condition dependence**	**Trigger period (cycles)**	**Duty cycle**	**Average percentage of incorrect outputs**
2	2:1	1, 2	cntd , cntf	800	1/(8)	12.5 %
4	2:1	2, 4	cntd , cntf	6400	1/(64)	3.125 %
8	2:1	3, 8	cntd , cntf	204,800	1/(2048)	0.1953 %
16	2:1	4, 16	cntd , cntf	104,857,600	1/(1,048,576)	0.00076%
32	2:1	5, 32	cntd , cntf	1.37 x 10^{13}	1/(1.37 x 10^{11})	1.164 x 10^{-8} %

designed in a way to make the trigger period as high as possible. However, the equations of duty cycle and percentage of incorrect outputs are independent of L.

3.4.2 Design of trigger combination circuits

Trigger combination circuits serve as the link between original control signals and the inputs of the multiplexers inserted as part of obfuscation. These circuits have to satisfy the following design requirements:

- Even if the key-gate muxes are recognized during inspection of netlist, the trigger combination circuits need to be indistinguishable from the original circuit making them difficult to detect and remove.
- The logic of the trigger combination circuit should have sufficient randomness and permutations to avoid traceability. This means that an attacker should not be able to trace from the key input upto the counter output successfully.
- The correct and incorrect signals need to be obfuscated similarly so these cannot be differentiated structurally.

With these design goals in mind, we create combination circuits using simple elements. Examples of fusion of control signals and trigger signals using different gates are shown in Fig. 3.13. In this figure, $C1$ is the original control signal and $C1'$ is its complement. Two versions of the trigger signal can be used: T and T'. All the example gate combinations shown can successfully combine the trigger and original signals of the design.

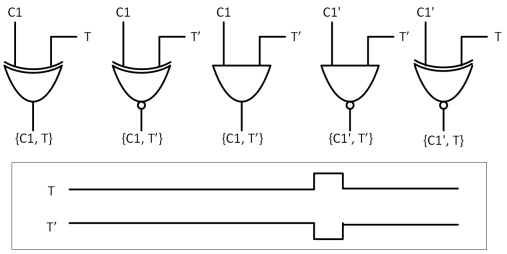


Figure 3.13: Logic gates of the trigger combination circuit to combine the generated trigger signals with existing control signals

The trigger combination circuit composed of logic gates is integrated into the controlpath of the mode based design as shown in Fig. 3.14. In this example, the key size is 8 bits and four 4-input muxes are required. Each mux requires one correct signal and three incorrect signals. A total of 12 trigger signals and 1 dummy signal are input to the trigger combination circuit. The dummy signal is used to make a dummy combination of the correct control signal with the trigger signal. The dummy signal may be all-1 or all-0 signal. This makes the correct control signal structurally similar to all other control signals.

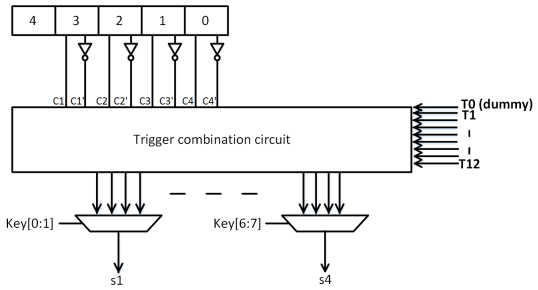


Figure 3.14: Trigger combination circuit added between derived control signals and the key-bit mapped multiplexers for a key size = 8 bits

We see that the trigger combination circuit is a key aspect of the security of the design. Complexity can be introduced into this block by using more logic gates and more levels between the control signals and mux inputs. This basically creates a layer of permutation between the counter outputs and the multiplexer.

3.5 Security Analysis

In a typical attack model for analyzing security of obfuscation schemes, it is assumed that the attacker has access to the obfuscated netlist which could be obtained by reverse engineering. It is also assumed that a functional IC with a correct working configuration and I/O pairs are available. The netlist provides information about the structure of the design and internal signal values while the functional IC provides correct I/O pairs. The additional components of the design including trigger circuit and trigger combination circuit are designed such that they are indistinguishable from the original circuit components. With these assumptions in mind we consider possible attacks and the security level of obfuscation as described in the next subsections.

Unlike fixed obfuscation, dynamic obfuscation has inherent time-dependence and randomness associated with it. Hence, these two factors play an important role in all attacks considered for dynamic obfuscation. For simplicity, we first consider the example of using 2 key bits as was illustrated in Fig. 3.2. General equations applicable to all key sizes are then provided. For all attacks except the SAT solver attack, we assume that every input pair applied to the system is able to distinguish between correct and

incorrect keys. This makes the attacks input independent and provides a lower bound for all security measures. In practical circuits, the security measure values will be higher, ensuring stronger security.

3.5.1 Lower bounds on security using brute-force attack

In an attack involving brute-force, various key values are input to the system and the output is verified to determine if the key is correct or not. Since we have made the worst case assumption that the system is input independent, we can assume that any input is capable of distinguishing between correct and incorrect key values. Let us consider the lower bounds on fixed, time-varying and dynamic obfuscation using these assumptions.

Fig. 3.15 illustrates the timing diagrams of fixed and time-varying obfuscation schemes using the example of 2 key bits as mapped in Fig. 3.2. For simplicity, the value of L is equal to 1 cycle. For fixed obfuscation, the outputs which are independent of time are shown. For time-varying obfuscation, occurrence of trigger in trigger signals is illustrated. For measuring a lower bound, we assume that if an attacker gets lucky, the first key value checked is the correct key. In the case of fixed obfuscation, the correct key can be determined in 1 cycle. In the case of time-varying obfuscation, the trigger signal occurs anywhere in the range of $t0$-$t1$ cycles, with a probability of 1. Hence, we need to monitor 2 cycles to confirm that the key is indeed correct.

Next, we consider the timing diagram of Fig. 3.12 for dynamic obfuscation. In this figure, cycles $t0 - t7$ need to be monitored to ensure no triggers occur for the correct key. Hence, we need to consider at least 8 cycles. The cycles to be monitored is equal to the trigger period (T) discussed earlier. Hence, using Equation (3.4), we obtain the following *lower* bounds on the security of the design:

$$time\_lower\_bound\ (fixed) = 1$$

$$time\_lower\_bound\ (time - varying) = LK$$

$$time\_lower\_bound\ (dynamic) = LK2^K \tag{3.7}$$

Thus, we observe that regardless of the input, the time-varying and random nature of dynamic obfuscation makes the time to attack exponentially dependent on the key size

of the design. This is in contrast to fixed obfuscation where time to attack is constant and time-varying obfuscation where there is a linear dependence on the key value.

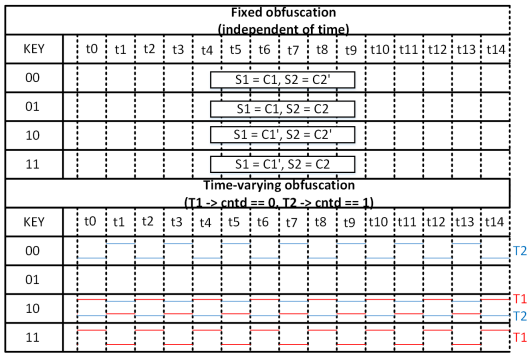


Figure 3.15: Timing diagram of corruption of outputs in fixed obfuscation and trigger signals in time-varying obfuscation

3.5.2 Average time to attack using brute-force

In this section, we calculate the average time to attack using brute-force. For a fixed obfuscation technique, on an average 2^{K-1} keys need to be tried. Note that the assumption of input independence is still valid. For dynamic obfuscation, this time is increased by the trigger period of the signal. For every incorrect key value, one or more trigger signals are selected. For observing a trigger with a probability of 1, time T is required. With two trigger signals, the probability of occurrence of trigger is increased and time to observe corruption is decreased to $\frac{T}{2}$. Hence, the time required to traverse through at least half the key values on an average is given by Equation (3.8). In this equation, t is the clock period of the original circuit. A detailed derivation of the average time to attack is provided in the Appendix B.

$$\begin{aligned} time\_brute\_force &= \sum_{n=1}^{K/m} (M-1)^n \frac{1}{n} \binom{\frac{K}{m}}{n} \frac{T}{2} t \\ &= \sum_{n=1}^{K} \frac{1}{n} \binom{K}{n} \frac{T}{2} t \ (using\ 2:1\ mux) \\ &= \sum_{n=1}^{\frac{K}{2}} (3)^n \frac{1}{n} \binom{\frac{K}{2}}{n} \frac{T}{2} t \ (using\ 4:1\ mux) \end{aligned} \tag{3.8}$$

Table 3.2: Brute-force attack time for different key values, trigger period and L with system operating at 100MHz

Key size	**L**	**Trigger period (cycles)**	**Time brute force attack**	**Effective key size**
2	100	800	10 us	11
4	100	6400	0.274 ms	16
8	100	204,800	0.0771 s	24
16	100	104,857,600	77.26 mins	40
24	100	53,687,091,200	9.35 yrs	56
32	100	1.37 x 10^{13}	605,222 yrs	72
2	1000	8000	100 us	14
4	1000	64,000	2.7 ms	19
8	1000	2,048,000	0.7711 s	27
16	1000	1,048,576,000	12.87 hrs	43
24	1000	536,870,912,000	93.55 yrs	59
32	1000	1.37 x 10^{14}	6,052,227 yrs	75

Using the equations for trigger period provided in Equation (3.4), configurations provided in Table 3.1 and different values of L, we tabulate values for the time to attack in Table 3.2. We assume a system operating at a frequency of 100 MHz, i.e., a clock period of 10 ns. This table also provides effective key size values which indicate the key size that would be needed if a fixed obfuscation technique were used. This value is calculated using $log_2(\frac{time\_brute\_force}{t}) + 1$. We observe that the effective key size is more than double of that of the actual key size. We also observe that as the value of L increases, the effective key size for the same trigger circuits and key size increases. This indicates the importance of choosing a rare event to increase the trigger period of the trigger signal.

3.5.3 Average time to attack using Reverse engineering

In this attack, we assume that information available from the netlist of the design and intermediate signal tapping can also be used is used to decipher the key. For example, in the case of a sequential circuit like FFT, if the control signals used for modification and multiplexer insertion are assumed to be known, a reverse engineering attack could be carried out. When an attacker tries to trace the key inputs of the design, it leads him/her to the key gates which are multiplexers in this case. The attack now reduces to finding correct select signals to obtain correct values at the outputs of the multiplexers.

For each multiplexer, one of the inputs is correct for the complete duration of operation of the circuit while the others have incorrect outputs for a short period of time. Thus, an attacker has to simulate the gate with inputs and different select signals and wait for a duration equal to the trigger period of the circuit. On an average, half the number of inputs of the multiplexer have to be checked to figure out the correct select signal. Repeating this process for all key-gates of the design results in time to attack as given by Equation (3.9).

$$\begin{aligned} time\_reverse\_attack &= \frac{num\_muxes \cdot size\_of\_mux \cdot T}{2} \\ &= \frac{\frac{K}{m} MT}{2} \\ &= \frac{K}{1}\frac{2}{2}T = KT \ (using\ 2:1\ mux) \\ &= \frac{K}{2}\frac{4}{2}T = KT \ (using\ 4:1\ mux) \end{aligned} \tag{3.9}$$

We observe that this value depends on the trigger period of the circuit and the number of key bits. From Equation (3.4), the trigger period has an exponential dependence on the key size. We use the same equations and configurations as done for the brute-force attack and tabulate our observations in Table 3.3. We observe that the time complexity of attack and effective key size are still large compared to the actual size of key inserted. This indicates that even though the time complexity decreases with respect to a brute-force attack, it is still a high value. Moreover, even though reverse engineering is a powerful attack, the assumptions involved may not be practical unless an attacker is equipped with expensive resources required for observation and measurement.

Table 3.3: Time complexity of reverse engineering attack at multiplexers for different key sizes and values of L

Key size	**No. of muxes**	**L**	**Trigger period(cycles)**	**Time to attack(cycles)**	**Eff. key**
2	2 2:1	100	800	1600	12
2	2 2:1	10000	80,000	160,000	18
4	4 2:1	100	6400	25,600	16
4	4 2:1	10000	640,000	2,560,000	22
8	8 2:1	100	204,800	1,638,400	22
8	8 2:1	10000	20,480,000	163,840,000	28
16	8 4:1	100	209,715,200	3.35×10^9	33
16	8 4:1	10000	2.09×10^{10}	3.35×10^{11}	39
24	12 4:1	100	8.05×10^{10}	1.93×10^{12}	42
24	12 4:1	10000	8.05×10^{12}	1.93×10^{14}	48
32	16 4:1	100	2.75×10^{13}	8.79×10^{14}	51
32	16 4:1	10000	2.75×10^{15}	8.79×10^{16}	57

3.5.4 SAT solver based attacks

SAT solver based attacks with respect to logic obfuscation schemes was introduced in [31]. In a SAT solver based attack, the obfuscated circuit is used as input to the SAT solver tool and it is assumed that correct I/O pairs from the functional IC are available. In each iteration, the SAT solver tries to find a distinguishing I/O pair which produces different outputs for different keys (for example, $K1$ and $K2$). The functional IC is then checked to verify which of the outputs is correct and the corresponding key is discarded. For all subsequent iterations, the SAT solver needs to ensure that all previously identified I/O pairs are also verified.

The time of execution of SAT attack is given in [42] and is reiterated here as Equation (3.10) .

$$time\_SAT\_attack = \sum_{i=1}^{\lambda} t_i \tag{3.10}$$

The complexity of SAT attack depends on two factors: the time to solve SAT formulation in each iteration, t_i, and the number of iterations, λ. To make the SAT attack infeasible, either the time to attack t_i or the number of iterations λ need to be increased. In dynamic obfuscation, we argue that the time to attack t_i is increased due to the time-varying and dynamic nature of the obfuscation technique.

Consider an example of using dynamic obfuscation with a 2 bit key as illustrated in Fig. 3.12. For an input pattern to be considered as distinguishing between key values 00 and 01, atleast 8 cycles should be monitored. This confirms that key 00 produces different outputs compared to key 01. After verification with the functional IC, key value 01 is discarded. In the next step, to check if key 00 and key 10 produce different outputs, 8 cycles need to be monitored. To check if key 10 satisfies the first distinguishing pattern, 4 cycles need to be monitored. When this fails, the next key value 11 is checked which again requires 8 cycles. Thus, finding the distinguishing input pattern has the same time-complexity as brute-force attack.

From Equation (3.8), the average time to perform a brute-force attack is dependent on the trigger period. T is in turn exponentially dependent on the key size. This forces the SAT solver tool to execute with exponential time complexity. A formal proof using experimentation is beyond the scope of this work since SAT solvers cannot be directly applied to sequential circuits.

Table 3.4: Gate counts of counter based trigger circuits for different key sizes

Key size	**Counter sizes {cntd, cntf}**	**Area (um^2)**	**Gate count**	**Power (uW)**	**Timing (ns)**
2	{1,2}	89.96	43	4.32	0.87
4	{2,4}	196.23	94	7.12	1.06
8	{3,8}	352.77	169	10.50	1.34
16	{4,16}	644.61	309	16.54	2.78
32	{5,32}	1221.48	587	28.22	3.48

3.6 Results

There are two sources of overhead associated with the proposed obfuscation techniques. First, we consider the overhead due to counter based trigger circuits. Second, we consider the overhead due to trigger combination circuits and the use of multiplexers for mapping of key bits. For the second overhead analysis, we make use of the discussed FFT example and report overheads with respect to its unobfuscated implementation. We also consider two other examples from the high level modules of ISCAS'89 benchmarks [43] and OpenCores platform [44]. The implementations are targeted towards ASIC design and Cadence technology library of 65 nM is used. We use Verilog HDL for modeling and Synopsis Design Compiler for synthesis. The designs are clocked at a frequency of $100MHz$. Area, Power and Timing of the designs as reported by the tool are tabulated. Gate counts are calculated using the Equation (3.11).

$$Gate\_count = \frac{Area\_of\_design}{Area\_of\_NAND\_cell} \tag{3.11}$$

3.6.1 Overhead of trigger circuits

For the area and power requirements of the trigger circuits based on counter design, we use configurations as suggested in Table 3.1. Key values in the range of 2 bits to 32 bits are considered. Tabulations of the area, gate count, power and timing are shown in Table 3.4. We observe that the counter circuits do not add significantly to both area and power overheads of the design. For most medium sized architectures, these counter

based trigger circuits could be inserted without varying the budget constraints of the device significantly. The low area and power also indicate that it is easy to maintain the stealthiness of these circuits.

3.6.2 Overhead of sequentially-triggered mode-based obfuscation and comparison with fixed-mode obfuscation

Example 1: 1024-point FFT circuit

For obfuscation, trigger based counter circuits are added using the configurations from Table 3.1. The 10-bit counter already existing in the controlpath is used for activation of the trigger circuit. The periodicity of this counter which is equal to 2^{10} is the activation rate L. We introduce keys of size 2, 4, 8, 16 and 30 bits using the 15 control signals indicated previously. We use either 2:1 mux or 4:1 mux depending on the number of key bits to be mapped. For example, a 2 bit key requires two 2-input muxes added to two control signals and two trigger signals. On the other hand, a 16 bit key can be added to the design by obfuscating 8 control signals with 4-input muxes.

We compare the overheads of the dynamic obfuscation with the fixed mode-based obfuscation scheme. For the fixed obfuscation scheme, we obfuscate the design using the same 15 control signals as in dynamic obfuscation. The two schemes are implemented using the same combination of key sizes and mux sizes and compared in Table 3.5. The table also presents the difference in security level in terms of time to attack. We observe that the time to attack for a key size of 30 bits is approximately 3.22×10^{21} cycles (1,021,055 years). A fixed obfuscation scheme of the same key size would be broken in just 536,870,912 cycles (5.36 s). The increase in security using a smaller key and resistance to other forms of attack come at a slightly higher cost with respect to area and power. However, the results reported are for the controlpath only (which constitutes 0.6% of the total design) and hence the overall area and power overheads are 0.32% and 0.07%, respectively.

Example 2: ISCAS benchmark - s298

The s298 circuit is a traffic light controller circuit generating patterns of red, yellow and green lights in a continuous sequence. It has several modes of operation: normal, fast

Table 3.5: Comparison of the overhead and time to attack of dynamic and fixed obfuscation for various sequential circuits with different activation rates L

Overhead of controlpath (0.6% of total) of a 1024-point folded, 2-parallel FFT circuit with L=1024							
		Dynamic mode of obfuscation			**Fixed mode of obfuscation [3]**		
Key	**Mux size**	**Area overhead**	**Power overhead**	**Brute-Force attack time (cycles)**	**Area overhead**	**Power overhead**	**Brute-Force attack time (cycles)**
2	2:1	3.77%	1.11%	10,240	0.19%	0.04%	2
4	2:1	7.50%	1.78%	281,258	0.43%	0.05%	8
8	2:1	13.96%	3.45%	78,959,021	0.97%	0.5 %	128
16	4:1	29.05%	6.45%	1.23 x 10^13	1.56%	0.78%	32,768
30	4:1	53.55%	12.41%	3.22 x 10^21	3.23%	1.22%	536,870,912
Overhead of Traffic light controller (s298) of ISCAS'89 sequential benchmark circuits with L=10							
		Dynamic mode of obfuscation			**Fixed mode of obfuscation [3]**		
Key	**Mux size**	**Area overhead**	**Power overhead**	**Brute-Force attack time (cycles)**	**Area overhead**	**Power overhead**	**Brute-Force attack time (cycles)**
2	2:1	24.50%	20.39%	100	2.33%	13.72%	2
4	2:1	49.24%	36.86%	2,746	4.08%	15.29%	8
8	2:1	62.66%	43.14%	771,084	7.70%	23.53%	128
Overhead of transmit module (1.7 % of total) of Ethernet IP core - OpenCores platform with L=65536							
		Dynamic mode of obfuscation			**Fixed mode of obfuscation [3]**		
Key	**Mux size**	**Area overhead**	**Power overhead**	**Brute-Force attack time (cycles)**	**Area overhead**	**Power overhead**	**Brute-Force attack time (cycles)**
2	2:1	3.31%	3.96%	655,360	0.18%	0.15%	2
4	2:1	8.41%	8.91%	18,000,554	0.23%	0.43%	8
8	2:1	16.12%	15.84%	5,053,377,350	3.16%	0.99%	128
16	2:1	29.11%	29.70%	3.04 x 10^14	4.82%	1.17%	32768
32	4:1	55.06%	46.53%	3.30 x 10^24	6.68%	2.97%	2,147,483,648

and blinking resulting in 8 different control signals and a mod 10 counter as control circuit. The functionality of the s298 benchmark circuit has been identified through reverse engineering and presented as high-level models [43]. We use this information to obtain the control signals necessary for obfuscation. In the obfuscated circuit, the sizes of the delay counter and the final counter are of size log_2K and K, respectively, where K represents the key size. It is to be noted that any tool capable of performing this type of reverse engineering, such as the method proposed in [30], will be able to identify sequential elements and derive control signals. We apply fixed and dynamic obfuscation to these signals using multiplexers and synthesize the resulting circuits to generate results shown in Table 3.5. Significant improvement in time to attack is observed compared to the fixed obfuscation, indicating the applicability of the proposed obfuscation techniques. However, since the circuit is small, the overheads are high.

Example 3: OpenCores - Ethernet IP core

The Ethernet IP core is one of OpenCore modules [44] and functions as a Media Access Controller (MAC) by connecting to the Ethernet PHY chip on one side and the Wishbone SoC bus on the other side. MAC Layer operations of transmitting, receiving, CRC generation and CRC check etc. at supported speeds of 10 and 100 Mbps bit rate are performed this IP core. Both the transmit and receive modules of this IP have control signals derived from counters. For example the transmit module has Nibble counters (16-bit), Byte counters (16-bit), CRC counters (3-bit) etc. which can be used for control flow modifications and dynamic obfuscation. In the obfuscated circuit, the sizes of the delay counter and the final counter are of size log_2K and K, respectively, where K represents the key size. The results of application of obfuscation to the transmit module are reported in Table 3.5. We observe significant improvement in security after application of dynamic obfuscation mainly because of the highly control-driven nature of the design and availability of large counters. Specifically, the time to attack is increased to 1,046,423,135 years with a key size of 32 bits compared to 21.47 s with fixed obfuscation. The overall overheads due to obfuscation are also observed to be 0.93% (area) and 0.79% (power). Hence, the proposed obfuscation is proved to be highly successful.

We have discussed three different examples of circuits on which the proposed obfuscation can be applied. We observe that the time to attack is significantly improved in

all cases. The overhead of obfuscation is high if the circuit size is small but remains constant even for larger circuit size, mainly dominated by the additional circuits incorporated for trigger generation and combination. Thus, for most practical circuits, the technique of obfuscation incurs low overheads. Finally, we observe that the availability of control flow information is necessary for the application of control flow obfuscation and its subsequent conversion to time-varying and dynamic obfuscation. Design documentation or tools which identify sequential elements [30] can be used to obtain this information.

3.7 Algorithm

- **Step 1: Selection of control signals for multiplexer insertion:**
  To identify control signals in the design, knowledge of the design or tools which identify sequential elements in the circuits [30] can be used.

- **Step 2: Selection of key size and multiplexer size:**
  Depending on the number of control signals C available for obfuscation, a key of size K can be mapped using select signals of size $m = K/C$ and multiplexers of size $M = 2^m$.

- **Step 3: Identification of activation circuit with period L:**
  Identify an activation part in the circuit (such as a counter used to derive control signals) with periodicity L.

- **Step 4: Design of trigger circuits and trigger combination circuits:**
  The size of delay counter ($size\_cntd$), size of final counter ($size\_cntf$) and the size of the random number generator are selected based on Equations (3.1),(3.2) and (3.3), respectively. The resulting values for trigger period, duty cycle and percentage of incorrect outputs are given by Equations (3.4), (3.5) and (3.6), respectively. Trigger combination circuits are designed to create a permutation layer between the multiplexers and control signals as described in Section IV.

- **Step 5: Measurement of security parameters:**
  Once the trigger circuits are designed, the selected parameters along with K, L

and M can be used to measure the various security values given by Equations (3.7), (3.8) and (3.9).

- **Step 6: Analysis of overheads due to obfuscation:**
  After addition of trigger circuit, trigger combination circuit and mapping of key bits to multiplexers, check if timing, area and power budget constraints are satisfied.

- **Step 7: Generation of obfuscated design:**
  Iterate through Steps 4-7 to obtain a final obfuscated circuit with desired level of security and acceptable overheads.

3.8 Dynamic obfuscation using two random number generators

The design of trigger circuits as presented in Section IV is just one among a possible set of configurations. In this section, we present an alternate design where instead of the final counter, another random number generator can be used as part of the trigger circuit as illustrated in Fig. 3.16. The advantage of using the second random number generator is the decrease in probability of occurrence of trigger which will in turn increase the lower bound on the time to brute-force attack as given by Equation (3.7).

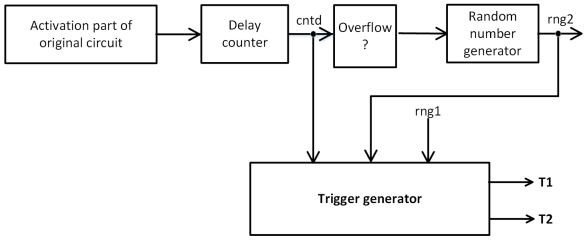


Figure 3.16: Structure of a sequential trigger generation circuit which uses two random number generators (rng1 and rng2) to generate triggers $T1$ and $T2$

When comparing the final counter (deterministic) to a random number generator (both of size K bits), the following probability is observed:

$$P(cntf = rng) = \frac{1}{2^K}$$

The final counter updates every LK cycles and hence the probability of occurrence of trigger is given by:

$$P(occurrence\ of\ trigger) = \frac{1}{LK2^K}$$

$LK2^K$ is the trigger period (T) for a 2:1 mux mapping. Hence, the probability of occurrence of trigger in one trigger period is 1.

Suppose the first random number generator is labeled $rng1$. Replace the deterministic final counter with another random number generator ($rng2$). Assuming $rng2$ is also of size K bits and the two random number generators are independent, the probability equation is changed to the following:

$$P(rng1 = rng2 = a) = P(rng1 = a)P(rng2 = a)$$

Similar to the final counter, $rng2$ updates every LK cycles since it is connected to the delay counter. Hence, the probability of occurrence of trigger is given by:

$$\begin{aligned} P(occurrence\ of\ trigger) &= \frac{1}{KL2^K}\frac{1}{2^K} \\ &= \frac{1}{KL2^{2K}} \end{aligned}$$

Therefore, during a trigger period $T = LK2^K$ the probability of occurrence of trigger is equal to $\frac{1}{2^K}$. Hence, the probability of occurrence of trigger is reduced to $\frac{1}{2^K}$ compared to the value of 1 if a deterministic counter is used. This results in an increase in the lower bound to brute-force attack given by:

$$time\_lower\_bound\ (dynamic) = LK2^{2K} \tag{3.12}$$

We limit our discussion in this work to the two configurations discussed and note that various other configurations can be used to design the trigger circuits. This will be addressed as part of future work.

3.9 Dynamic obfuscation using true random number generators

A simplification of the technique of dynamic obfuscation can be obtained by replacing the synchronous trigger generation circuit using counters with an asynchronous true

random number generator (TRNG). The new trigger generation circuit is presented in Fig. 3.17.

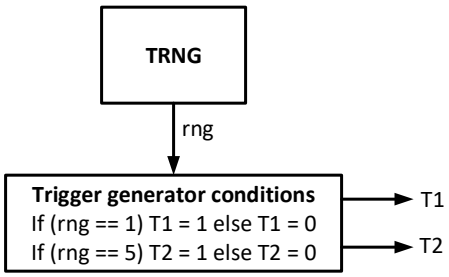


Figure 3.17: Structure of an asynchronous trigger generation circuit which uses a true random number generator (TRNG) to generate triggers $T1$ and $T2$

The source of randomness is a TRNG and the condition for trigger generation is based on comparison of the TRNG output (rng) with a specific value. Different conditions of comparison can be used to generate different trigger signals. The advantage of using a TRNG instead of a pseudo random number generator lies in the fact that TRNG derives randomness from the properties of the circuit itself and cannot be replicated. Hence, TRNGs are more secure and the asynchronous design greatly simplifies the design of trigger generation circuits.

Based on the new design, parameters of the dynamic obfuscation are redefined and security parameters are recalculated. Since no activation circuits or counters are required, only the size of the random number generator needs to be defined. Each trigger signal should be based on a different comparison test of the random number generator. Hence, the size of the random number generator is equal to the number of trigger signals required. From Section 3.4.1, the number of trigger signals is equal to $\frac{K}{m}(M-1)$ where K is the key size in bits, M is the size of the multiplexer and $m = log_2 M$. Thus, the

lower bound on the size of the TRNG is given by:

$$\begin{aligned} size\_trng &= log_2(\frac{K}{m}(M-1)) \\ &= size\_cntd \\ &\approx log_2 K(using\ 2:1\ mux) \\ &\approx log_2 K + 1(using\ 4:1\ mux) \end{aligned} \tag{3.13}$$

We observe that the size of the TRNG should be greater than or equal to the size of the delay counter of the synchronous trigger generation circuit. Also, note that in this design, even if multiple key values select the same trigger signal, there is no overlap since the trigger condition is based on a random number generator and the same comparison condition will be met at different time instants.

Assuming the TRNG generates random numbers following a uniform random distribution, the probability of occurrence of a particular random number during one period of operation of the TRNG is 1. Thus, the trigger period of the trigger generation circuit is now defined as:

$$\begin{aligned} trigger\_period(T) &= K(using\ 2:1\ mux) \\ &= 2K(using\ 4:1\ mux) \end{aligned} \tag{3.14}$$

Choosing a value of the TRNG equal to the minimum required value results in a trigger period which is not exponentially dependent on the key and hence too small. In order to bring an exponential dependence on the key size, the size of TRNG can be chosen equal to the key size K which results in a new value for the trigger period given by:

$$trigger\_period = 2^K \tag{3.15}$$

The duty cycle of the trigger signal is defined as:

$$\begin{aligned} duty\_cycle &= \frac{1}{trigger\_period} \\ &= \frac{1}{2^K} \end{aligned} \tag{3.16}$$

The average percentage of incorrect outputs can be derived by replacing the value

of the *duty_cycle* in the derivations of Section 3.11 to obtain:

$$\begin{aligned} avg\_incorrect\_outputs &= \frac{K}{2.2^K}(using\ 2:1\ mux) \\ &= \frac{1}{2^{2K}}\sum_{n=1}^{\frac{K}{2}} n3^n \binom{\frac{K}{2}}{n}(using\ 4:1\ mux) \end{aligned} \tag{3.17}$$

For different key sizes, the trigger period, duty cycle and average incorrect outputs are tabulated in Table 3.6.

Table 3.6: Trigger period, duty cycle and percentage incorrect outputs for different key sizes and $size\_trng = K$

Key size	**Mux size**	**TRNG size**	**Trigger period (cycles)**	**Duty cycle**	**Average percentage of incorrect outputs**
2	2:1	2	4	1/(4)	25%
4	2:1	4	16	1/(16)	12.5%
8	2:1	8	64	1/(64)	1.562%
16	2:1	16	65536	1/(65536)	0.012%
32	2:1	32	4294967296	1/(4294967296)	3.725×10^{-7}%

Next, the measures of security including lower bound on time to attack, brute-force attack time and reverse engineering attack time are recalculated. The lower bound on time to attack assumes that the correct key has been entered in the very first attempt. The attacker still needs to check for a duration of time equal to:

$$time\_lower\_bound = trigger\_period = 2^K \tag{3.18}$$

For brute force attack time and reverse engineering attack time, the same equations as Equation (3.8) and Equation (3.9) can be used. In these equations, the value of T needs to be replaced by 2^K. Using these measures, the different attack times are tabulated in Table 3.7. For a key size of 32 bits, the brute-force attack time is equal to 189 years. We observe that the brute-force attack time is reduced for the asynchronous design using a TRNG when compared to using a synchronous design using counters. However, the value is still much higher compared to the brute-force attack of fixed obfuscation which is equal to 21.47 s.

Table 3.7: Brute-force attack time and reverse engineering attack time for different key sizes K

Key	**Mux size**	**Brute-force attack time (cycles)**	**Reverse engineering attack time (cycles)**
2	2:1	5	8
4	2:1	69	64
8	2:1	9.638×10^{03}	2048
16	2:1	2.897×10^{08}	1048576
32	2:1	5.964×10^{17}	1.3744×10^{11}

Note that since all the measures are now completely dependent on the size of the random number generator, as the size is increased, stronger security can be obtained. However, this results in greater area and power consumption due to the requirement of a larger true random number generation circuit. Hence, there is a trade-off between security and area/power overhead for this design. This design would be most useful if a system already contains a true random generator resulting in no additional overheads.

3.10 Conclusions

This chapter presents a dynamic obfuscation technique and its algorithm as summarized in Section VII. We demonstrated that this scheme offers a simple yet effective solution for successfully obfuscating systems. We also presented security analysis in terms of time complexity of attack and overheads in terms of area, power and timing. A comparison of the new method with traditional multiplexer based methods was also discussed. Also, the proposed obfuscation technique can be used to hierarchically obfuscate designs similar to the hierarchical mode-based obfuscation approach presented in [45].

Though the ideas presented here are demonstrated on sequential circuits only, these techniques are equally applicable to combinational circuits. For insertion of multiplexers, nodes not susceptible to fault sensitization can be used [27]. Future research should

be directed towards design of a computer aided design tool that can automatically generate obfuscated designs from original designs. A formal proof of security against SAT based attacks is also a direction of future work. Whether model checkers can be used to reduce the time to attack is another topic of future research. Finally, future research should be directed towards understanding whether side channel attacks can be used to learn the correct keys and reduce time to attack.

3.11 Proof of average percentage of incorrect outputs per key

Consider the mapping of key bits to a M:1 mux as shown in Fig. 3.10 and let M equal to 2. For the first multiplexer, an incorrect value of key bit produces corruption at output S_1 for *duty_cycle* period of time. From Equation (3.5), this value is given by $\frac{1}{K2^K}$. Thus, if a key value selects an incorrect value only for one key bit, outputs are incorrect for $\frac{1}{K2^K}$ of the time. If a key value has incorrect values for key bits of first and second multiplexer, then outputs are incorrect for $2 \times \frac{1}{K2^K}$ of the time and so on.

Considering all key combinations and corresponding number of incorrect key bits selected by them, the total duration of time for which corruption in output occurs for all keys can be given by:

$$total\_incorrect\_outputs = \sum_{n=1}^{K} n\binom{K}{n}\frac{1}{K2^K}$$

On average, the corruption in outputs produced by each key is given by:

$$avg\_incorrect\_outputs = \frac{\sum_{n=1}^{K} n\binom{K}{n}\frac{1}{K2^K}}{2^K}$$

Using the identity $\sum_{n=1}^{K} n\binom{K}{n} = K2^{K-1}$, the equation simplifies to:

$$\begin{aligned} avg\_incorrect\_outputs &= \frac{K2^{K-1}\frac{1}{K2^K}}{2^K} \\ &= \frac{1}{2^{K+1}} \end{aligned}$$

Thus, the average percentage of incorrect outputs is given by Equation (3.6).

Similarly, consider the case where $M = 4$. Each multiplexer is mapped to two key bits ($m = 2$). Hence, there are three possible incorrect key combinations and 1 correct

key combination at each multiplexer. If a key value selects incorrect key bits for only the first mux, the total corruption in outputs is given by $3 \times duty\_cycle$. If a key value selects incorrect key bits for the first and second mux, there are $3 \times 3 = 9$ incorrect key combinations and hence, the total corruption in outputs is given by $9 \times duty\_cycle$ and so on.

For a 4:1 mux mapping, the $duty\_cycle$ is given by $\frac{1}{K2^{K+1}}$ from Equation (3.5). The number of multiplexers are $\frac{K}{m} = \frac{K}{2}$. Considering all key combinations as for 2:1 mux mapping but considering 3^n key combinations for every n multiplexers selected, we get the total corruption of outputs as follows:

$$total\_incorrect\_outputs = \sum_{n=1}^{\frac{K}{2}} 3^n n \binom{\frac{K}{2}}{n} \frac{1}{K2^{K+1}}$$

Dividing by the total number of keys, we get the average incorrect outputs as:

$$\begin{aligned} avg\_incorrect\_outputs &= \frac{\sum_{n=1}^{\frac{K}{2}} 3^n n \binom{\frac{K}{2}}{n} \frac{1}{K2^{K+1}}}{2^K} \\ &= \frac{1}{2K} \sum_{n=1}^{\frac{K}{2}} n 3^n \binom{\frac{K}{2}}{n} \end{aligned}$$

3.12 Proof of average time to attack using brute force

If one trigger signal is selected by a key value, the probability of occurrence of trigger in one trigger period is 1. Therefore, the time to observe a trigger is equal to trigger period T. The probability of occurrence of trigger is increased to 2 if two trigger signals are selected by a key. Thus, the time to observe one trigger is reduced to $T/2$. If n trigger signals are selected, the time of observation of one trigger is reduced to T/n. The maximum number of trigger signals that can be selected by each key is equal to key size K bits.

The number of incorrect key combinations for both 2:1 and 4:1 muxes are the same as considered in Appendix A. Specifically, for a 2:1 mux this is given by $\sum_{n=1}^{K} \binom{K}{n}$ and for 4:1 mux it is given by $\sum_{n=1}^{\frac{K}{2}} 3^n \binom{\frac{K}{2}}{n}$. Using these equations and the observation of

reduction in time to observe trigger, we get the following equation for total time to brute force attack using 2:1 mux mapping:

$$total\_time\_brute\_force = \sum_{n=1}^{K} \binom{K}{n} \frac{T}{n}$$

On an average, only half the keys need to be explored in a brute-force attack. Also, each cycle is assumed to operate with a time period t. Using these, we obtain the following equation:

$$time\_brute\_force = \frac{\sum_{n=1}^{K} \frac{1}{n} \binom{K}{n}}{2} Tt$$

For a 4:1 mux mapping, the following equation is obtained:

$$time\_brute\_force = \frac{\sum_{n=1}^{\frac{K}{2}} \frac{3^n}{n} \binom{\frac{K}{2}}{n}}{2} Tt$$

Generalizing these equations to key bits mapped to any mux size $M = 2^m$, we obtain the equation given by Equation (3.8).

Chapter 4

Comparative study of Authenticated Encryption

Today's smart, connected devices pose a complex challenge to cryptographic designers in terms of better resource efficiency, area constraints, flexibility and robustness. In this chapter, we look at some new Authenticated Encryption (AE) schemes from an ongoing Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR), and evaluate their benefits to lightweight applications such as sensor networks, RFID, IPSec, Internet-of-Things, implantable medical devices and wearables. We summarize candidates after extensive studies of their properties such as nonce misuse resistance, security measures, parallelizabilty and existing hardware and software implementations, and recommend a few candidates most suitable for different applications.

4.1 Introduction

Authenticated encryption schemes are a class of symmetric-key cryptographic algorithms that simultaneously provide both confidentiality and authenticity of data. Confidentiality involves protecting data from being disclosed without authorization, while authenticity encompasses ensuring both integrity of data and verification of its source. Some applications require the handling of additional data such as packet headers, which demand authentication without the need for any encryption. Such schemes are broadly

classified as Authenticated Encryption with Associated Data (AEAD) and will largely be the focus of this work. A typical interface for AEAD, is shown in Fig. 4.1 and can be described using the equations 4.1 and 4.2.

$$Enc : E_k(N, AD, P) \Rightarrow \{C, T\} \tag{4.1}$$

$$Dec : D_k(N, AD, C, T) \Rightarrow \{P, \perp\} \tag{4.2}$$

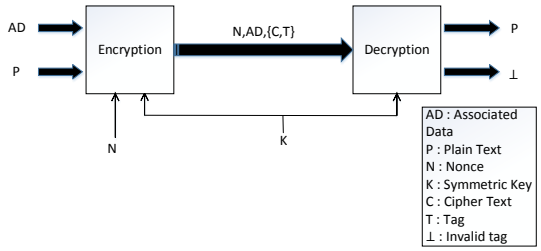


Figure 4.1: General scheme of an Authenticated Encryption scheme with Associated Data

It is possible to build a generic composition for authenticated encryption schemes, combining an encryption scheme which exhibits indistinguishability under chosen plaintext attack (IND-CPA) and Message Authentication Codes (MAC) which are unforgeable [46]. However, these approaches are not efficient both with respect to performance and resource usage. Unless implemented correctly, these compositions tend to be error-prone as has been observed in practice. For example, padding oracle attacks have broken the security of MAC-then-Encrypt schemes, while using the message directly for tag calculation in Encrypt-and-MAC schemes makes the confidentiality of the message questionable. Moreover, authentication of associated data provides an additional challenge when trying to combine the two goals of security as outlined in [47]. Hence, the need for a secure and efficient authenticated scheme, continues to be a challenge for today's designers.

While many proposals for AEAD have been presented over the last decade, the encryption schemes that have gained popularity include the GCM(Galois/Counter mode) [48], CCM(Counter with CBC-MAC) [49], EAX [50] and OCB(Offset Codebook) [51] modes.

The AES-GCM uses AES in the counter mode (CTR) and a Galois mode of authentication. Several advantages including ease of implementation have made AES-GCM to be popularly adopted in various applications.

In this work, we look at AEAD schemes from a perspective of usage in lightweight applications. In this regard, we first study the architecture of AES-GCM and then look into some new and efficiently constructed schemes from an ongoing competition CAESAR. This work also serves as a comparison between the different candidates of CAESAR with AES-GCM as reference. An overview of this kind, considering functional and architectural aspects, performance measures, comparison with AES-GCM and an application oriented discussion does not exist in current literature and will be highly beneficial for designers of cryptographic protocols for embedded system platforms and SoCs.

4.2 AES-GCM

AES built in the Counter (CTR) mode of operation and hashing over the Galois field are combined to obtain AES-GCM authenticated encryption. Offering several advantages, such as high speeds at low cost and low latency, parallelizability and efficient software implementations with table driven operations, the AES-GCM has been widely researched architecturally. Several implementations exist in literature serving both efficiency and high performance [52, 53]. A simplified architectural representation of AES-GCM is provided in Fig. 4.2., to demonstrate its ease of implementation in hardware.

Even though AES-GCM is the most widely adopted algorithm, a few shortcomings have been observed in the algorithm in recent literature. Message forgery attacks have been demonstrated on the polynomial hashing used for authentication based on weak keys [54]. Also, the requirement of a unique nonce for each key is cumbersome and requires careful implementation and good practices from users and is found to be seldom followed in practice [55]. Moreover, the algorithm still suffers from having to use Galois field multipliers for authentication and there is a precomputation overhead for every new key input into the design.

Changing scenarios of applications of AEAD algorithms has necessitated the need to revisit AES-GCM and explore its advantages or shortcomings from this new paradigm.

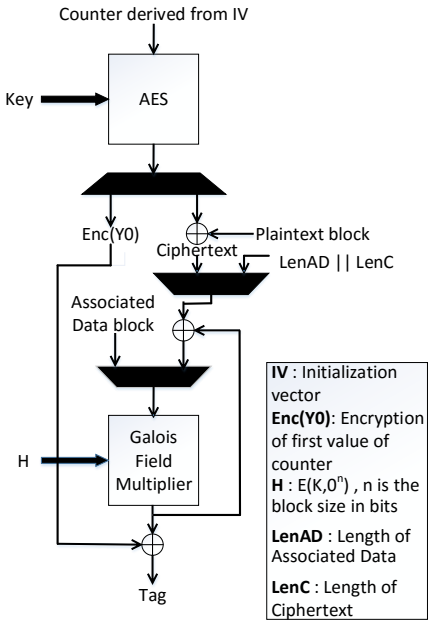


Figure 4.2: Simplified block diagram of AES-GCM using minimal resources

With security becoming indispensable in several areas of IoT, we need to look into implementations to consider the suitability of algorithms in reference to these devices. For example, could an extremely lightweight protocol be built for resource constrained devices? Could security be given a higher importance in comparison to performance? Would the energy efficiency and availability constraints be met? Are the algorithms flexible enough to provide for updation of parameter values when necessary? Hence, a need to broaden the scope of study of authenticated encryption schemes beyond AES-GCM has risen, paving the way for new areas of research.

4.3 CAESAR competition

CAESAR is a competition for Authenticated Encryption: Security, Applicability and Robustness, aimed at looking for new and improved versions of AEAD schemes, possibly overcoming the shortcomings of AES-GCM algorithm [56]. The competition began in 2014 and is currently in the third round, with 29 candidates selected for the 2nd round, from 54 initial submissions. The call for submissions puts forward the goals of a new

AEAD algorithm presented in the general format of an AEAD encryption scheme as described in Fig.4.1. The nonce in this case however, has been termed public message number and/or secret message numbers.

A comprehensive and easy to grasp summary of the first round of CAESAR candidates has been provided in [57], highlighting important features and categorizing them based on the underlying primitives used. In this work, we update this overview to consider only the second round candidates and include software and hardware implementations from various publications and tools existing in current literature. We also highlight some architectural details of each of the candidates and bring out important features. We intend the usage of this article as a reference for designers looking for new AEAD schemes more suitable to their applications without having to go through overwhelming number of papers. As a further step, we use these data summaries to select candidates for lightweight implementation scenarios discussed previously.

4.3.1 Primitives

For simplicity, we categorize the different submissions into three groups : block-cipher based, sponge/duplex based and others. These categories are based on the primitives that have been used to construct the candidates. Since, traditionally block ciphers have been used to construct AEAD, we see many submissions from this category. Many of them use AES or AES-like constructions while some of them build their own block cipher constructions. The next category is based on duplex constructions mostly using MonkeyDuplex construction derived from the permutation based sponge function from the new SHA3 standard, Keccak [58]. In the last category, we have grouped other types such as stream cipher based, permutation based and fiestel network based schemes.

4.3.2 Features

We now discuss some of the features of these candidates mostly from an architectural and application oriented perspective.

Nonce Misuse Resistance

Though the issue of nonce misuse has been debated over in several forums, with many accepting and others denying the issue, it is favorable for an algorithm to provide a nonce misuse resistant feature. Specifically in the case of IoT and like devices, storing and managing fresh nonces for each new message and key combination is going to become a challenging task. This will lead to error prone and incorrect implementation scenarios leading to security breaches. The nonce for the candidates has been split into a public message number(PMN) and secret message number(SMN). While the PMN behaves just like a typical nonce, the SMN is embedded inside the ciphertext and must be recoverable from it [59].

A few of the algorithms provide complete nonce misuse resistance in terms of both confidentiality and integrity, while a few provide for integrity with no confidentiality assured. Some of the algorithms use the SMN to guarantee security assuming different SMNs are used even when PMNs are repeated. While it is unclear how practical these categories maybe, providing for partial nonce misuse resistance can be considered a significant improvement over having none at all. We categorize the candidates based on their ability to provide nonce misuse resistance in a complete sense, partially or none at all. The summary for the same has been provided in Table. 4.1.

Security

Security constraints for algorithms need to be evaluated in terms of both strong mathematical proofs and thorough cryptanalysis by external groups. Such a deep level of security analysis is beyond the scope of this work. However, we summarize some of the security numbers as presented in [57], picking out values specific to a key size of 128 bits (as a more common case scenario) when the option is available. If not, other closer numbers such as 120 bits or 256 bits have been picked. The numbers are in terms of time complexity t and query complexity q, quantifying confidentiality and integrity. For our own analysis we consider that security of atleast 128 bits as acceptable for most lightweight industrial applications.

Table 4.1: Comparison of candidates based on nonce misuse resistance, security and parallelizability

Based on primitives	**Nonce-Misuse resistance**	**Candidates**	**Security in terms of time complexity (t) and query complexity (q)** **Key = 128 bits**	**Candidates**	**Parallelizability**	**Candidates**
Block cipher	Complete	Joltik, **Deoxys**, POET, AES-COPA, ELmD, **AEZ**, SHELL	t <128 bits q <64 bits	Joltik, SHELL	Yes	AES-COPA, **AES-OTR**, **AEZ**, **Deoxys**, ELmD, Joltik, Scream, Shell
	Partial	**CLOC**, **SILC**, **AES-JAMBU**	t = 128 bits q = 64 bits	ELmD, Scream, **CLOC**, **Deoxys**, **SILC**, POET, AES-COPA, **AES-JAMBU**	Partial	**CLOC**, **SILC**, POET
	None	Scream, **AES-OTR**	t =128 bits q >64 bits	**AES-OTR**, **AEZ**	No	**AES-JAMBU**
Sponge/Duplex	Complete	Primate-APE	t<128 bits q<64 bits		Yes	Icepole, **Keyak**, **NORX**, StriBOB
	Partial	Pi-Cipher(SMN based), Icepole(SMN based), **Keyak**, **NORX**	t=128 bits q=64 bits	**Ascon**, **NORX**	Partial	
	None	**Ascon**, Primate-HANUMAN, Primate-GIBBON, StriBOB, **Ketje**	t=128 bits q>64 bits	PRIMATEs, Pi-Cipher, StriBOB, Icepole, **Keyak**, **Ketje**	No	**Ascon**, **Ketje**, PRIMATEs
Others (Stream, Permutation, Feistel)	Complete	Paeq, HS1-SIV, Minalpher	t<128 bits q<64 bits		Yes	**Acorn**,**Aegis**, Paeq, Pi-Cipher, **Tiaoxin**, Trivia, Minalpher
	Partial	**Acorn**	t=128 bits q=64 bits	**Acorn**, **Aegis**	Partial	
	None	Trivia, **MORUS**, **Aegis**, OMD, **Tiaoxin**	t=128 bits q>64 bits	Trivia, **MORUS**, Paeq, HS1-SIV, OMD, Minalpher, **Tiaoxin**	No	HS1-SIV, **MORUS**, OMD

Parallelizability

Algorithms which are parallel can help both software and hardware designers in implementing better designs through the use of various optimization techniques known in literature. Hence, the feature of whether an algorithm is parallelizable or not, is useful in determining if a resource optimized implementation could be obtained. We also categorize the candidates based on whether they are inherently parallel or not and if they provide some form of partial parallelizability. This means that even though the algorithms are not fully parallel with respect to a single message, when multiple messages are handled, the algorithm can be constructed in a parallel fashion. This is presented in Table. 4.1.

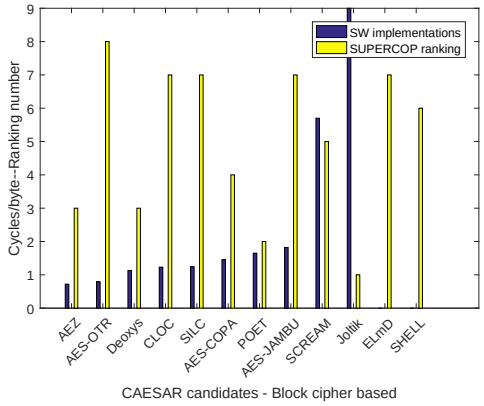
4.4 Performance analysis

4.4.1 Software implementations

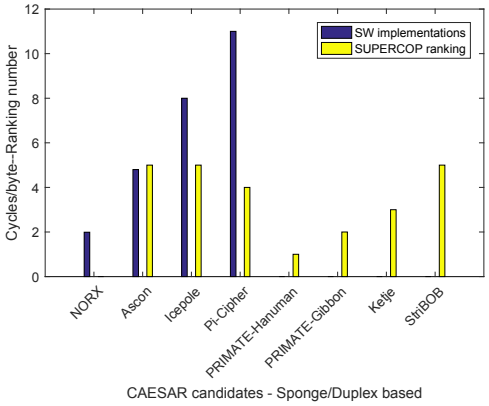
The measurement of performance with respect to a candidates software implementation, is annotated in terms of cycles per byte(cpb). As part of Round 1 and Round 2 submissions, software implementations have been submitted by each of the candidates. Most of these have been implemented on the latest processors including Intel Haswell or Sandy Bridge, employing well known AES NI instruction sets and SIMD instructions [60], [61], [62], [63], [64], [65], [66], [67], [68]. Apart from these implementations, we have also considered the implementations from SUPERCOP benchmarking tool [69], well presented in [70], for Intel Core i7-4770 and Intel Core i5-3210M. These resources have helped us rank the candidates in terms of software performance as represented in Fig.4.3. For custom implementations, cpb is reported and for benchmark implementations, a ranking number which provides relative speed is shown.

4.4.2 Hardware implementations

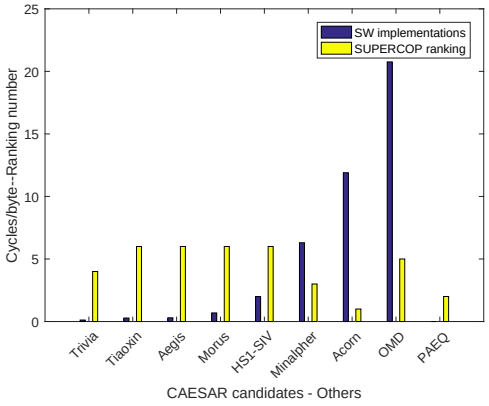
The difficulty of benchmarking hardware arises due to the existence of a varying range of hardware platforms. This includes families of FPGA's such as Xilinx which use LUTs for measurement and Altera which uses LEs, different CAD tools performing different type of optimizations giving varying synthesis results, ASIC libraries etc. In this regard,



(a) Block-cipher



(b) Sponge/Duplex



(c) Others

Figure 4.3: Software performance numbers based on candidate implementations and SUPERCOP benchmarks

the Cryptographic Engineering Research Group(CERG), GMU is playing a major role in proposing a hardware API and releasing a benchmarking tool ATHENa [71]. Existing implementations from this tool and other hardware implementations from various existing publications have been considered for this analysis.

Even though we have performance numbers in terms of area and throughput for both FPGA and ASIC implementations, we consider FPGA implementations to be more reliable. ASIC implementations provide the area performance in terms of gate counts which are completely dependent on process, technology and libraries used. The existing implementations [72], [73], [74], [75] for FPGA are available for different generations of Xilinx and Altera FPGAs, necessitating the need for normalization which has been done by taking reference implementations of AES-GCM on these same or similar platforms from the ATHENa website. Similar to the rankings for software performance, we have prepared charts to rank hardware performance in terms of area, throughput and throughput/area as illustrated in Figure 4.4. These easy to grasp graphs give a good idea of algorithms which are lightweight or speed oriented and were used by us for our further analysis as well.

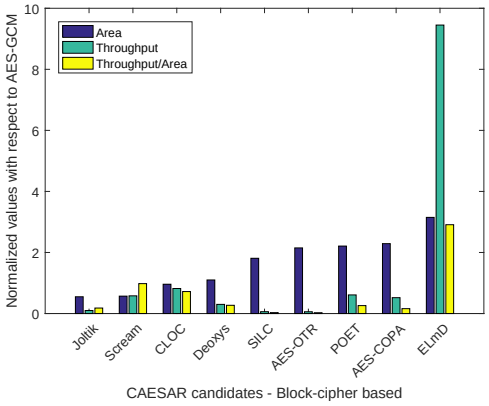
4.4.3 Memory footprint

Most candidates focus on software performance with respect to cpb and hardware performance with respect to area and throughput. However, it is of paramount importance for lightweight applications to look into their memory footprint with respect to embedded system implementations. We researched a few protocols in this regard and found performance numbers for the same. We summarize this in Table. 4.2.

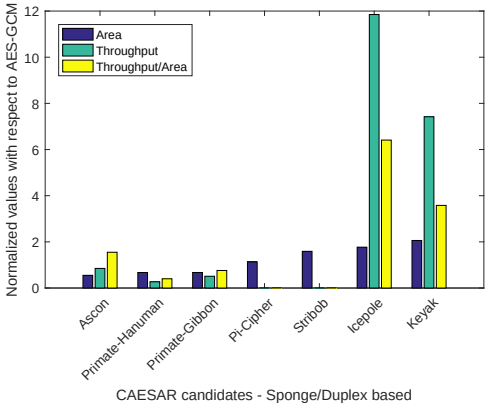
4.5 Summary with recommendations of candidates

4.5.1 Block-cipher based

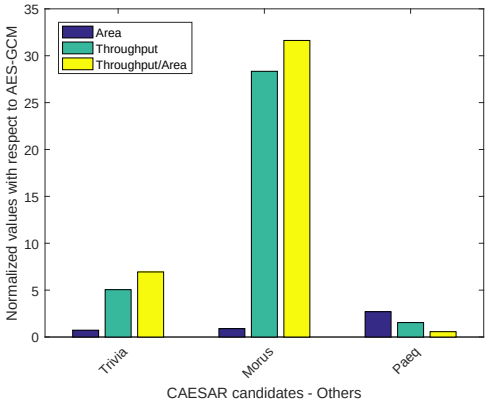
Joltik and **Deoxys** [56] are AEAD schemes built from custom made tweakable block ciphers (TBC) and use linear and lightweight transformations for their top module. They show excellent hardware and software performances and are available in both nonce misuse resistant and non-resistant modes. While Deoxys provides full security



(a) Block-cipher



(b) Sponge/Duplex



(c) Others

Figure 4.4: Hardware performance numbers based on candidate implementations and ATHENa tools

Table 4.2: Candidates with implementations on embedded platforms and associated memory footprints

Candidate	**Device**	**ROM (bytes)**	**RAM (bytes)**
CLOC	ATmega128	2980	362
SCREAM	Atmel AVR	6442	160
WhirlBOB	Cortex-A8	608	-
Minalpher	RL78 microcontroller	510	214

beyond birthday bound, Joltik has lower security at lower implementation cost. These algorithms have no precomputation overhead or long initialization.

SCREAM [56] also makes use of a custom built Tweakable authenticated encryption block(TAE) providing security beyond the birthday bound and masking to achieve security against side channel attacks such as Differential Power analysis and Electro-Magnetic analysis. Being a highly parallel architecture, allows SCREAM to exploit SIMD instructions providing for good software implementations and efficient hardware architectures. However, Scream does not provide any kind of nonce misuse resistance.

CLOC and **SILC** [56] use AES as the underlying block cipher and provide partial nonce misuse resistance with respect to integrity, while providing no guarantees on confidentiality. They handle short data very well by having minimal precomputation and low memory requirements. They fare decently well in terms of both software and hardware performances and provide acceptable levels of security.

POET [56] is another well rounded candidate based on AES and providing features such as complete nonce misuse resistance and parallelizability. These algorithms are targeted at low latency, high throughput applications with large data. While the scheme is well pipelineable and offers a birthday bound on security, the requirement of inverse operation of decryption may serve as a caveat.

AEZ [56] implements an encode-then-encipher mode of operation providing for strong security and complete nonce misuse resistance. The architecture automatically exploits novelty and redundancy in messages, giving good software implementation performance. However, this architecture suffers from being extremely difficult to implement

in hardware.

Table 4.3: Summary of candidates highlighting properties and possible application scenarios

Algorithms	Summary of properties	Application scenario
DEOXYS, MORUS, ASCON	Excellent performance characteristics both in hardware (throughput) and software (cpb)	Wireless sensor nodes, IPsec
POET, AEZ, DEOXYS	Provide high security (complexity of attack) and complete nonce misuse resistance	Automotive IoTs, Defense related
JOLTIK, CLOC/SILC, ASCON, PRIMATEs, PI-CIPHER, SCREAM	Implementations with low area, power and memory footprint requirements	RFID, smart cards, wearables, Implantable medical devices
DEOXYS, AEZ, POET, JOLTIK, PRIMATE-APE, CLOC/SILC	Complete or partial nonce misuse resistance of ensuring guaranteed authentication	All applications

4.5.2 Sponge/Duplex based

Based on the Spongewrap/MonkeyDuplex mode of operation and eliminating inverse operations, **ASCON** [56] presents a low hardware scheme with extremely low memory requirements. While the scheme provides full security of 128 bits, it offers no nonce misuse resistance.

The **PRIMATEs** [56] come in 3 flavors and have varying goals of lightweight (HANUMAN), High speed (GIBBON) and additional security (APE). While these schemes have shown good hardware and software performances, and provide high security numbers, only PRIMATE-APE provides complete nonce misuse resistance.

This **Pi-Cipher** [56] architecture uses an Addition, Rotation and XOR (ARX) based function and a parallel, incremental architecture most suitable for long messages. Its main advantage is that it does not use the inverse operation. While providing for intermediate tags, these schemes also provide partial nonce misuse resistance in terms of SMN as described before. Simple operations of its underlying function, results in good hardware performances while providing for decent software speeds and high security.

4.5.3 Others

While there are many schemes in this category, **MORUS** [56] which uses a dedicated structure that has both hardware and software performance benefits by making use of simple operations such as shift, and, xor and SIMD instructions, respectively. Excellent

security is provided in terms of both confidentiality and integrity. However, no nonce misuse resistance is claimed.

Lightweight applications such as smart card, RFID, etc. demand low area and low memory footprint. AEAD schemes suitable for implementation in wearables additionally require that the power consumed and energy per bit of processing be minimal. On the other hand, the security protocols of industrial wireless sensor nodes demand high performance in software or hardware depending on the implementation used. Parallelizability allows for efficient implementations which can be beneficial to protocols implemented in IPsec. Security critical applications such as automotive IoTs demand higher levels of security in terms complexity of attack and complete nonce-misuse resistance. Finally for all lightweight applications, some level of nonce-misuse resistance is highly desirable since not having to generate and maintain unique nonces will result in lower resource requirements. The candidates discussed are suitable for one or more applications and the specific application to which they can be associated is listed in Table 4.3. An attempt to identify and categorize candidates according to use cases is also being undertaken as part of Round 3 of CAESAR competition.

4.6 Conclusions and future work

We have provided a comprehensive review of the competition and parameters to be evaluated with respect to considering new AEAD schemes for applications. In Table 4.3, we tabulate the discussion of our selected candidates and provide a good reference for future users of the algorithms. Similar analysis can be carried out for other candidates, following the discussions in this work. Also, we understand that no analysis is complete without a thorough power and energy efficiency analysis. This is the future scope of our research while we try to closely look into the architectural aspects of these candidates and provide more insights.

Chapter 5

Nonce-Misuse Resistant Authenticated Encryption Architectures

This chapter presents a performance comparison of new *authenticated encryption* algorithms which are aimed at providing better security and resource efficiency compared to existing standards. Specifically, these algorithms improve the security of existing authenticated encryption standards by providing a critical property termed *nonce-misuse resistance*. The work addresses algorithm to architectural mappings of several candidates from the ongoing Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) as well as a submission from the Crypto Forum Research Group (CFRG). Implementations of the architectures on both field programmable gate arrays (FPGA) and application specific integrated circuits (ASIC) platforms are provided and compared with the architecture of a popular standard AES-GCM (Advanced Encryption Standard in Galois Counter mode). Optimizations which are applicable to authenticated encryption in general and nonce-misuse resistant architectures in particular are also presented. A hardware-software co-design approach to optimization is also discussed. The implementations via proposed optimizations demonstrate that new authenticated encryption algorithms can provide comparable performance as standard AES-GCM while enhancing security and resource utilization for specific use-case

scenarios.

5.1 Introduction

With the advent of the Internet of Things (IoT) era, billions of devices will be connected to each other and to a common network. Hence, it is of utmost importance to ensure security and privacy of communication between the devices as well as between the device and the cloud server. Moreover, there is a growing trend to bring computing to the edge of the IoT rather than the cloud, termed as edge-centric computing [1]. Hence, resource efficient and strongly secure cryptographic algorithms which can ensure security of communication and can be implemented on the hardware of the device itself have become critical. Also, algorithms which require smaller key sizes, less frequent change of keys and better resilience are desired.

Authenticated Encryption (AE) algorithms combine the process of authentication and encryption to create a single algorithm which is secure and resource efficient. It is well understood that confidentiality of data in itself does not suffice and it is important to ensure authentication of source as well as data integrity [46]. Hence, AE algorithms are growing in importance with the changing device scenarios and platforms. When data such as packet headers, message numbers etc. are also included as part of the plain-text, the resulting algorithms are termed Authenticated Encryption with Associated Data (AEAD) algorithms. These additional data require only authentication and are termed associated data.

The general equations of AEAD are expressed as follows:

$$Enc_K(N, AD, PT) = (CT, Tag) \tag{5.1}$$

$$Dec_K(N, AD, CT, Tag) = (PT, \perp) \tag{5.2}$$

In these equations, K represents the secret key, AD represents the associated data, PT refers to the plain-text and N is a unique non-repeating number termed *nonce*. These inputs are applied to the encryption algorithm Enc to produce the outputs, CT and Tag. The output of encryption of plaintext is CT (cipher text) and Tag is utilized for authentication purposes. The nonce is used to transmit more than one message block using the same secret key. For the decryption algorithm Dec, N, AD, CT, Tag and

K are used as inputs to retrieve PT. Additionally, Tag is verified and a valid/invalid message is generated (represented as $\perp$ in the equation).

Some of the existing standards for AEAD algorithms include Advanced Encryption Standard in Galois Counter Mode (AES-GCM), Advanced Encryption Standard in Counter with CBC-MAC mode (AES-CCM), EAX, Offset Codebook mode (OCB) etc. We do not delve into the details of these algorithms and refer the reader to existing literature [50, 51, 76, 77]. Instead we focus on AES-GCM which has been deployed in most practical applications such as Transport Layer Security (TLS), Secure Socket Layer (SSL) etc. Using AES-GCM as a baseline architecture, we present the architectures of several new algorithms which have been proposed as part of the ongoing CAESAR competition as well as from CFRG. The chosen algorithms incorporate an important property termed *nonce-misuse resistance*.

The presented nonce-misuse resistant algorithms are mapped to hardware using both FPGA and ASIC platforms. The implementations are then compared to the AES-GCM standard. This work also discusses important architectural differences that arise as a consequence of the nonce-misuse resistance property and optimizations to overcome these limitations. FPGA based implementation of two of the algorithms discussed in this work, namely Deoxys and AES-GCM-SIV have been presented in [8] and [9], respectively. This work moves beyond these implementations by providing several optimizations and comparisons with other candidates such as PRIMATE-APE and POET. Also, a hardware-software co-design approach to obtain resource efficient implementations applicable to modern IoT like platforms is presented. A final comparison of the selected candidates and the standard both with respect to performance as well as resource utilization are presented with a recommendation of the most suitable candidates for several use-case scenarios.

5.2 Motivation and Related work

In this section, we discuss the shortcomings of the current standard and motivation for this work. The algorithm and architecture of AES-GCM are also briefly discussed.

5.2.1 New AEAD schemes

Several issues have been identified in existing authenticated encryption algorithms. Some of these are highlighted below:

- Existing algorithms are still be too large in terms of area or consume too much energy. This is specially true if the authenticated encryption schemes are to be implemented on a device which has minimal resources.
- Several security properties such as nonce-misuse resistance, decryption misuse resistance, robustness against leakage of plaintext, detection of forgery attempts etc. are desirable and missing in existing AEAD algorithms.
- Cryptanalytical efforts have found groups of weak keys in the most widely adopted AES-GCM algorithm [78].
- Better performance while maintaining same level of security of existing standards or better security with the same performance are both desirable especially because of the increase in number of devices and reduction in the size of devices in modern applications.

In an attempt to alleviate some of these problems, a call for novel authenticated algorithms was put forth in the form of CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness). The goal of this competition is to identify a portfolio of authenticated encryption algorithms which offer advantages over AES-GCM and are suitable for widespread adoption [79]. The competition is ongoing and currently in its final round with 7 finalists. The first round had 54 submissions out of which 29 candidates were selected for the second round. In the third round 15 potential candidates were recognized. These candidates have different properties with respect to security, resource consumption, underlying constructions, etc. A summary for the candidates and their important properties can be found in [7, 80]. A candidate that we discuss in this work, Deoxys, has been selected to the final round of the competition. Note that even though some of the candidates selected for this study have not been selected for the final round of the CAESAR competition, these are useful candidates for several applications. Since the competition considers several parameters apart from

security, these candidates have been excluded from the next round. We also consider an algorithm submitted to CFRG after the CAESAR competition first round submission deadline.

Several independent implementations of the algorithms in both hardware and software can be found in literature. In an attempt to bring these implementations under the same platform, there is an ongoing effort being carried out by the SUPERCOP software benchmarking team and the ATHENa GMU hardware platform team [81]. Our work specifically focuses on in depth analysis of nonce-misuse resistance schemes whose details and significance will be discussed next.

5.2.2 Importance of nonce-misuse resistance

From the description of AEAD schemes, it is observed that nonces are critical to the security of symmetric key cryptographic algorithms. The construction of these algorithms is such that using the same key, multiple messages can be encrypted by using different nonces. Hence, the nonce must not repeat under the same key. Even though this appears to be a simple requirement, it is not easy to satisfy as has been observed in many practical applications [55].

There are several approaches that can be used to ensure nonces do not repeat while using the same key. First, one solution is to update the keys frequently. However, regular exchange of secret keys between two parties is not an easy process and many applications will not have the capability to do so. Specifically, with systems involving IoT devices where millions of devices are interconnected with each other, the most practical implementation would program one secret key and utilize it for the lifetime of the device. The second approach to ensure nonces do not repeat is to derive the nonces from counters. With sufficiently large counters, the nonce values will be based on each increment of the counter and will not repeat. However, if the counter is made to overflow by injecting faults or other forms of attacks, the nonces start to repeat breaking the security of the algorithm. Finally, the nonce can be made unique by using random number generators. However, the random number generator should be of high quality and sufficiently large. Ensuring this requirement is met is again a challenging task with the ever shrinking sizes of devices.

Attacks due to nonce-misuse have been demonstrated in literature in important applications such as the Transport Layer Security (TLS) application [55]. Thus, algorithms which can inherently provide some form of nonce-misuse resistance have become favorable and will continue to increase in importance as IoT devices become more prevalent.

5.2.3 Description of AES-GCM

The AES-GCM algorithm is represented using the block diagram of Fig. 5.1. This algorithm is a block cipher based authenticated encryption scheme. This implies that a block cipher such as Advanced Encryption Standard (AES) is used to perform all encryption operations. The authentication is handled by a Galois field multiplier which performs polynomial multiplication. It is to be noted that the algorithm descriptions presented in this work mainly discuss the flow of data with respect to associated data and plain-text/message blocks. The data flow is then translated to architectures which consume less power and energy while providing acceptable performance. The data-flow of AES-GCM indicates that each plain-text block is first processed by AES to generate a cipher-text block. The cipher-text block is then processed by the multiplier. After all message blocks are processed, the tag is generated. Note that the nonce in this case is a counter whose value is incremented for each message block.

5.2.4 Architecture of AES-GCM

We map the algorithm presented Section 5.2.3 to a serial implementation aimed at low area and power consumption. This means that the architecture makes use of a single AES block and Galois Field Multiplier block. The blocks of the datapath of AES-GCM are illustrated in Fig. 5.2. The plain-text blocks are processed serially and require correct control to direct data in and out of the blocks. All architectures created in this work follow the serial implementations. Based on the different properties of the architecture, optimizations are then added on top of the serial architecture. This is discussed in Section 5.5.

The finite state machines required for control are presented in Fig. 5.3. Note that after processing of a ciphertext block by the encryption block, the first state machine generates a *ct_done* signal. The authentication FSM first processes the associated data

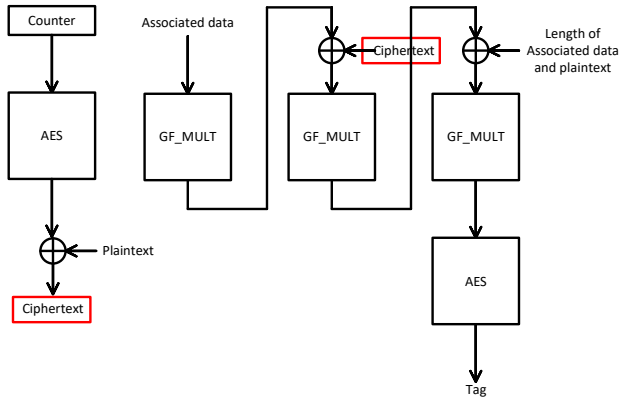


Figure 5.1: Description of the algorithm of AES-GCM. The left part of the figure presents ciphertext block generation using a counter. The right part of the figure presents associated data processing and processing of plaintext blocks to generate the tag.

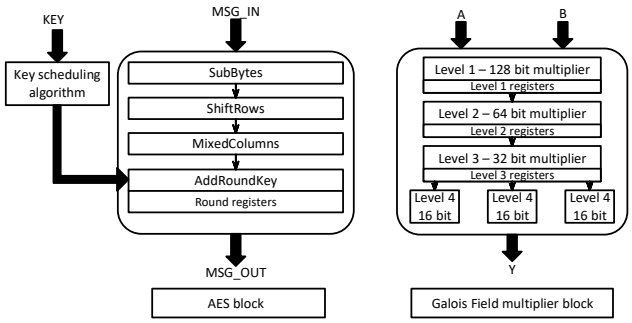


Figure 5.2: Data-path of AES-GCM consisting of AES block and Galois Field multiplier block.

blocks and waits for the *ct_done* signal. Upon receiving the signal, the processing of ciphertext blocks is performed in an interleaved manner with the encryption process to finally generate the tag value. The ability to parallely process message blocks is an important property of the AES-GCM algorithm since it results in good performance and resource utilization for AES-GCM.

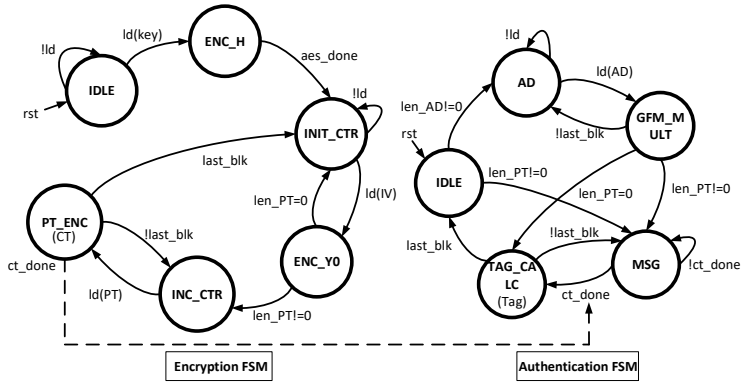


Figure 5.3: Finite State Machines (FSM) of AES-GCM for encryption and authentication in a block-interleaved manner. Signal ct_done is used for synchronization between the two FSMs.

5.3 Algorithms with nonce-misuse resistance

Nonce-misuse resistant schemes ensure that even though the nonce is repeated, only limited knowledge about the encrypted message is given away, while the complete plaintext decryption is not possible [82]. There are two types of nonce-misuse resistant algorithms: *complete* and *partially* nonce-misuse resistant. While the complete nonce-misuse resistance property is preferred, some applications may still benefit from partial nonce-misuse resistance. Thus, we consider both types of algorithms in this work. The algorithms considered are from the CAESAR competition and other independent proposals.

Table 5.1 lists the candidates that have been selected for our study. The description of the basic properties of these candidates with a special emphasis on the nonce-misuse resistance property are provided in the subsequent subsections. A more detailed description of all candidates with comparison of properties can be found in [7, 80].

5.3.1 Deoxys

Deoxys is a block-cipher based authenticated encryption algorithm utilizing a tweakable block cipher, *Deoxys-BC*. The tweakable block cipher is based on AES but uses a key and a tweak value. It has more rounds than standard AES, claiming better security.

Table 5.1: Summary of candidate features and comparison with AES-GCM

Parameters	AES-GCM	Deoxys	AES-GCM-SIV	POET	Primate-APE
Nonce-misuse resistance	No	Complete	Complete	Complete	Complete
Security level	t = 64 bits q = 64 bits	t = 128 bits q = 64 bits	t = 64 bits q = 64 bits	t = 128 bits q = 64 bits	t = 128 bits q >64 bits
Parallelizable	Yes	Yes	Yes	Partial	No
Comparison to AES-GCM	-	- Authentication first - Secure block cipher	- Similar components - Authentication first	- AES for both Auth & Enc - Flexible	- Lowest resource - Slower

*t=time complexity, q=query complexity

**The detailed security analysis of AES-GCM-SIV based on nonce-repetition frequency and message block length is beyond the scope of this work and can be referred from [83, 84]. [1]

The Deoxys algorithm has two modes: one for which nonce must not be reused and one which is nonce-misuse resistant. The nonce-misuse resistant version of Deoxys provides full 128-bit security for unique nonces and birthday bound security when nonce is reused. Existing hardware and software implementation also show that it is suitable for short messages having low pre-computation overhead.

The basic steps of Deoxys authentication and encryption are described in Fig. 5.4. Since the block cipher in Deoxys is a custom built tweakable block cipher, the inputs include an additional tweak value which needs to be provided each time the encryption block is called. From the block diagram, we observe that the associated data blocks are processed first followed by the plain-text blocks. Finally, incorporating the nonce value into the tweak value, the tag is generated. Note that for the encryption process, the nonce value is used as input to the block cipher. The tag value is incorporated in the tweak value. Thus, there is a dependence between the tag generation and cipher-text generation in Deoxys. This will lead to important differences in the architectural implementation and optimizations as will be seen in Section 5.4 and Section 5.5.

5.3.2 AES-GCM-SIV

AES-GCM-SIV combines the Galois Counter mode (GCM) building blocks into a Synthetic IV (SIV) paradigm [85]. Thus, it ensures complete nonce-misuse resistance. The basic steps of encryption and decryption are shown in Fig. 5.5. The main advantage of AES-GCM-SIV is its similarity in structure to AES-GCM algorithm. Mainly, the AES

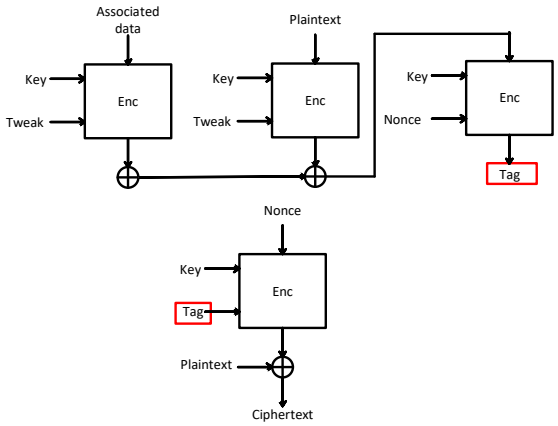


Figure 5.4: Description of the algorithm of Deoxys. The encryption block in this figure is the tweakable Deoxys block cipher with key and tweak as inputs. The top part of the figure presents tag generation. The generated tag is used for processing of plaintext blocks to generate ciphertext blocks.

block cipher is still used in CTR mode in this algorithm. The POLYVAL construction used for building the multiplication is similar to the GHASH construction used in AES-GCM. The conversion between the two is expressed as Equation (5.3).

$$POLYVAL(H, X_1, X_2, .) = ByteReverse(GHASH(By \\ teReverse(H) * x, ByteReverse(X_1), ByteReverse(X_2), .)) \quad (5.3)$$

However, there are several differences in the overall construction of the algorithms. The tag generation occurs first in AES-GCM-SIV compared to the generation of cipher-text blocks occurring first in the case of AES-GCM. Secondly, the plain-text needs to be processed twice in the case of AES-GCM-SIV instead of just once as in the case of AES-GCM. The occurrence of tag generation before cipher-text and the processing of plain-text twice is common to both AES-GCM-SIV and Deoxys algorithms. We make an assumption that the plain-text is stored in memory and can be accessed by the algorithms twice. Future studies will address the practicality and limitations of such an assumption.

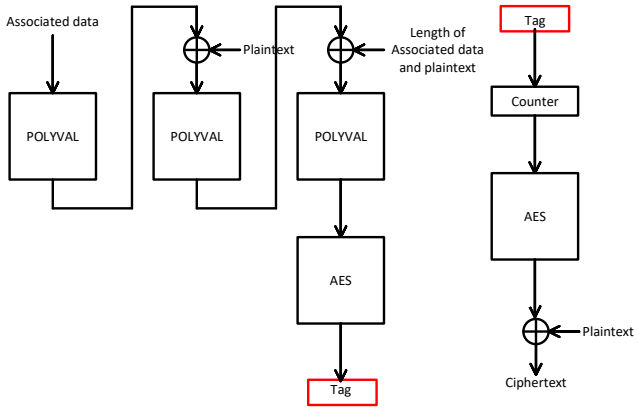


Figure 5.5: Algorithm description of AES-GCM-SIV. The left part of the figure represents processing of associated data and message blocks using the POLYVAL and AES block to produce a Tag. The right half of the figure represents ciphertext generation using the Tag as an initial value for the counter.

5.3.3 POET

POET is a block-cipher based algorithm which provides both nonce misuse resistance as well as decryption misuse resistance. This means that if authentication fails, the adversary obtains no information about decrypted cipher-texts. POET uses the AES block cipher itself for both encryption and authentication and does not require any Galois field multiplication operations. Thus, the algorithm can reuse available AES modules and provide required authenticated encryption operations. The POET algorithm is also described as a flexible algorithm since it is not necessary to use AES block ciphers in this construction. It is shown that the AES can be replaced with a smaller 4-round AES block cipher or other such lightweight block ciphers. This will reduce the resource overhead of POET while providing lower security which can be suitable for some applications. The authors also propose an encryption only version of the algorithm termed POE whose details can be referenced from [86].

The top-level algorithm description of POET is provided in Fig. 5.6. The associated data blocks are processed using a hashing key L and the AES block cipher blocks to produce an intermediate value τ. This is represented in the first part of the figure. The processing of plain-text blocks occurs next. This occurs through the AES block

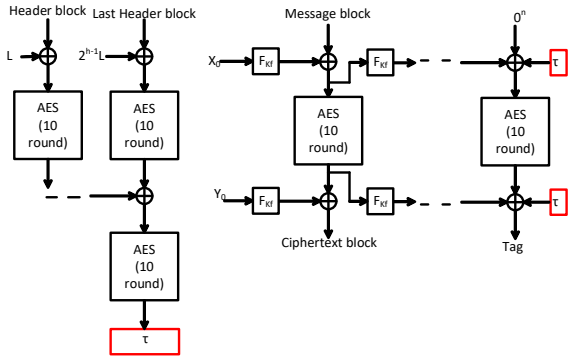


Figure 5.6: Algorithm description of POET. The left part of the figure presents processing of associated data/header blocks to produce an intermediate value τ. Ciphertext and Tag generation using the intermediate τ value are presented in the right half of this figure.

ciphers and hash functions F_{K_F}. Here we note that the output of the message processing blocks is used together with the value τ to generate the final tag. Hence, in POET, the cipher-text generation occurs first similar to AES-GCM algorithm.

5.3.4 PRIMATE-APE

Unlike the block-cipher based algorithms, the PRIMATE-APE is a sponge/duplex based construction termed permutation-based authenticated encryption. The sponge construction is based on a fixed-length permutation block which operates on a fixed number of bits. These bits are divided into two parts: r also known as the bit-rate or rate and c termed as capacity. The r bits of the input are absorbed by multiple permutation function blocks in an iterative manner and squeezed out in subsequent stages. The capacity bits contribute to the permutations but are never changed or released as an output. Using sponge constructions in different ways, both ciphers and hashing algorithms can be created. The duplex construction is a variation of the sponge construction in that absorption and squeezing of data occur in an alternating fashion.

The PRIMATE-APE algorithm utilizes a duplex based construction to create an authenticated encryption algorithm. The permutation block is termed as PRIMATE in this algorithm. A high level overview of the construction is presented in Fig. 5.7.

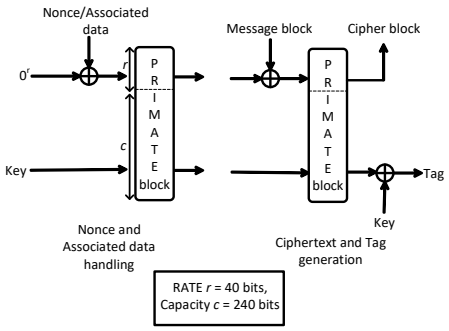


Figure 5.7: Algorithm description of PRIMATE-APE. After initial absorption of the nonce and associated data by the PRIMATE block, for every message block that is input, the cipher text block is produced. Finally, the tag is generated.

We observe from this figure that the nonce/associated data blocks are absorbed in an iterative manner by the permutation block followed by the absorption of the message blocks. Finally the cipher-text block and tag are generated from the last permutation block. The inputs at the top which form the rate r are of size 40 bits and the input at the bottom which forms the capacity c is of size 240 bits. Hence, a 240-bit key is required and processing of associated data/message blocks occur in blocks of 40 bits each. The details of the PRIMATE block are discussed in Section 5.4.4.

5.4 Architectural descriptions

In this section we discuss the architectural descriptions of the selected candidates in detail. Note that the architecture of AES-GCM was presented in Section 5.2.4. For the nonce-misuse resistant algorithms under study, we describe blocks which are either different from or additionally required compared to AES-GCM. The focus of this section is to highlight these differences with respect to the standard to make it easier to comprehend and evaluate the additional resources required for implementations of these new algorithms.

5.4.1 Deoxys

The Deoxys algorithm uses the Deoxys Block cipher for both authentication and encryption purposes. Thus, a hardware implementation requires only one block cipher as illustrated in Fig. 5.8. For the implementation of the block cipher, we use a standard implementation as used in AES. Since the rounds of Deoxys and AES are similar, the implementation is also similar. However, Deoxys has more number of rounds (14 rounds) compared to 10 rounds of AES. The key schedule is also different in the case of Deoxys and is based on the tweakey schedule.

In this work, we consider two versions of the Deoxys algorithm presented in literature namely the version v1.3 [87] and v1.41 [88]. One of the main differences between these two versions is the implementation of the tweakey key schedule algorithm. Briefly, Deoxys v1.3 requires the implementation of simple substitution function h and a multiplication function g. Deoxys version v1.41 replaces the multiplication function g with simple LFSRs. More details about the implementation can be found in the algorithm description of these two versions.

The state machine illustrated in Fig. 5.8 is used to route data to and from the block cipher to perform encryption and authentication process. Since the same block cipher is used for all operations, the implementation is completely serial in nature. Note that there are no multipliers required for this architecture unlike AES-GCM which reduces resource requirement and eliminates the problem associated with security of polynomial multiplication blocks.

5.4.2 AES-GCM-SIV

The architecture of AES-GCM-SIV consists of the AES block as well as the POLYVAL block. While the AES block is realized similarly to the AES block used in AES-GCM, the POLYVAL block is realized by making appropriate swap and shift operations to the GHASH composition of AES-GCM algorithm. This is described in detail in [85] and illustrated in Fig. 5.9. The components marked in red indicate the additional operations required for POLYVAL. The control-path of AES-GCM-SIV is also illustrated in this figure. Note that due to the dependence of the ciphertext generation on the tag generation process, only a single serial finite state machine just as in Deoxys can be

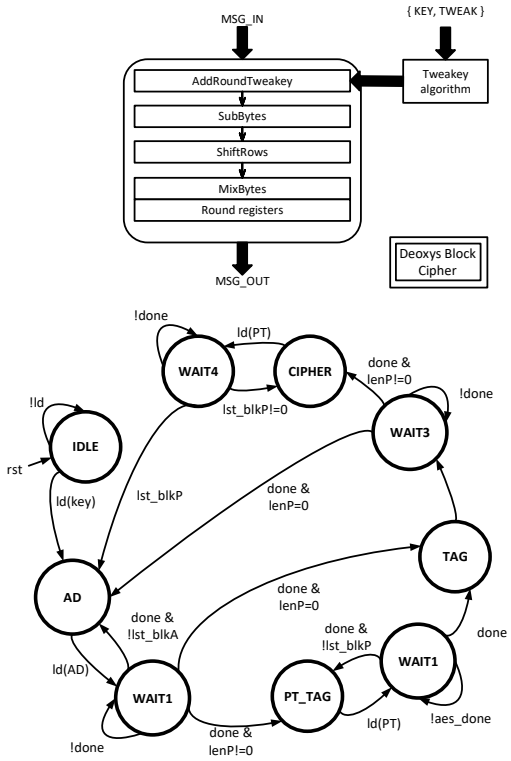


Figure 5.8: Datapath and Controlpath of the Deoxys algorithm. Note that the block cipher is a tweakable block cipher and a single state machine is used for both authentication and encryption.

used. The state machine routes data to and from the AES and POLYVAL blocks.

5.4.3 POET

The top level block diagram of POET is illustrated in Fig. 5.10. The architecture is constructed using one AES block cipher (which is utilized for both encryption and authentication) and two 4-round AES blocks which serve as the hash function (F_{K_f}) blocks. We observe from the algorithm description of POET (Fig. 5.6) that an initial vector X is first processed by the F_{K_f} block and xored with the message block. After processing of message block using the AES module, the Y vector processed by the F_{K_F} block is utilized to produce the cipher-text block. However, the processing of the next message block has a dependency on the F_{K_F} output of the previous message

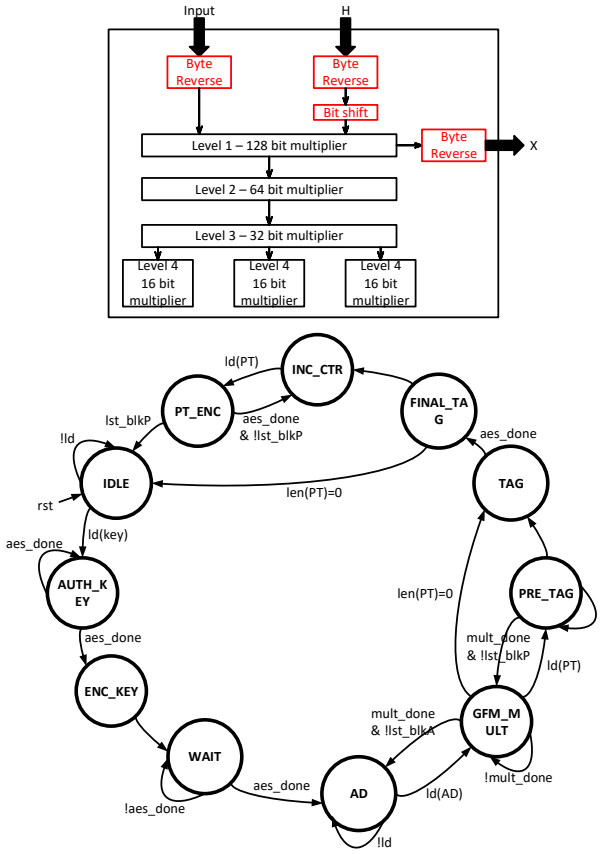


Figure 5.9: Data-path and control-path of AES-GCM-SIV. Only the conversion of *GF_MULT* of AES-GCM to *POLYVAL* of AES-GCM-SIV is presented here. Also note that a single FSM is used for both authentication and encryption.

block. Hence, the processing of Y vector and processing of previous message blocks is performed simultaneously using two F_{K_F} function blocks. Hence, the architecture of POET consists of two 4 round AES blocks as illustrated in Fig. 5.10.

5.4.4 PRIMATE-APE

The architecture of PRIMATE-APE mainly consists of the PRIMATE block as illustrated in Fig. 5.11. The PRIMATE block used in this work is a version termed PRIMATE-120 as described in [89]. The structure of PRIMATE is similar to the AES

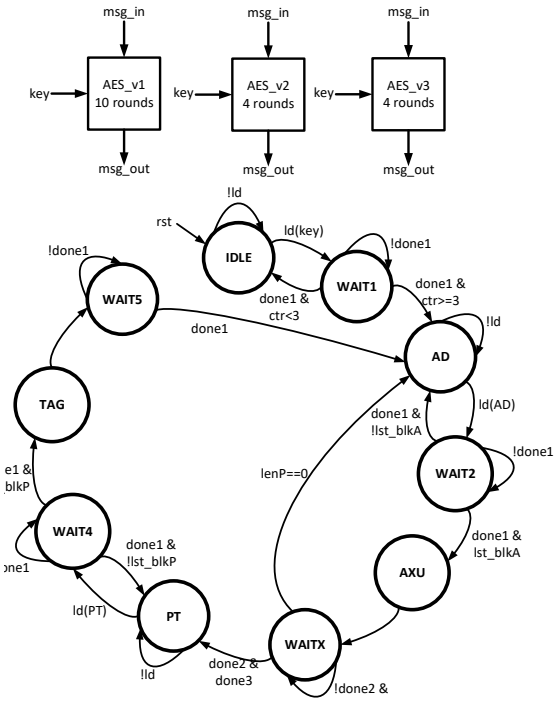


Figure 5.10: Datapath and Controlpath of the POET algorithm. Note that the datapath consists of a regular AES 10 round cipher and two AES 4 round ciphers used as hash function blocks.

block cipher and consists of rounds: SubElements, ShiftRows, MixColumns and ConstantAddition. Hence, the same architecture as used for AES in the block cipher based schemes described previously can be used. However, the state elements of PRIMATE are 5-bit elements arranged in a 7x8 state matrix. The first row of the state is the rate and the rest is the capacity. Also key schedule algorithm is not required in case of PRIMATE. 5-bit LFSRs are used to create the constants required for ConstantAddition round.

We observe from the architecture that the PRIMATE-APE is a lightweight authenticated encryption algorithm because of the use of only one PRIMATE block. However, it can also be observed that all transactions with this block occur through the rate bits which are only 40 bit wide. Hence, this architecture will be significantly slower than

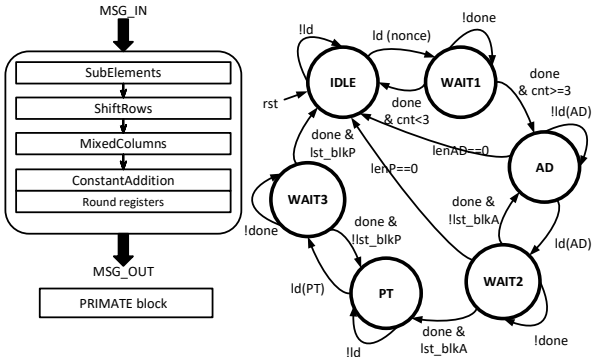


Figure 5.11: Datapath and Controlpath of PRIMATE-APE algorithm. Note that the implementation of the PRIMATE block is similar to AES block but is much smaller and utilizes nibbles for processing.

other architectures presented in this work. However, for short messages, this architecture will serve as a good lightweight option providing nonce-misuse resistance.

5.5 Optimizations

In this section, we discuss some of the optimizations that can be applied to the discussed AEAD schemes. These optimizations are either targeted to utilize the properties of the FPGA platforms on which the algorithms are implemented or the properties of the algorithm itself. In Section 5.7, we present the results of optimization on the algorithms and discuss which of the optimizations are most suitable for each algorithm.

5.5.1 Parallel processing of messages

From the implementation of AES-GCM we have observed that the AES-GCM is inherently parallel in nature. This means that when a cipher text block is being generated, parallel processing of data by the multiplication block can occur. This is an important property which results in low cycles per byte for the AES-GCM algorithm compared to other algorithms. However, by modifications such as pipelining and parallelism, this property can be utilized to optimize other algorithms. We discuss the optimization as applied to each algorithm in this section.

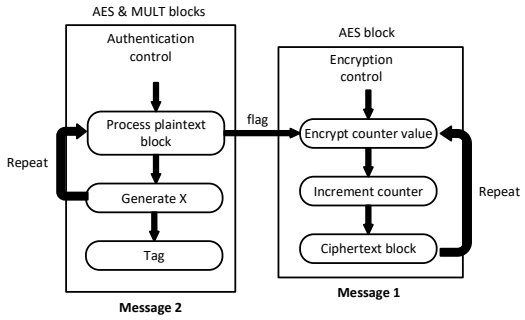


Figure 5.12: Parallel processing of messages in AES-GCM-SIV.

Application to AES-GCM-SIV

AES-GCM-SIV utilizes both AES and multiplier blocks for encryption and authentication operations. Hence, similar to AES-GCM, the two operations can be separated and performed in parallel. However, there is a data dependency between the authentication and encryption process. This means that first the authentication process has to be performed completely to generate the tag and then the cipher-text blocks can be generated through encryption process. To break this dependency, we can process two messages in parallel and pipeline the design such that the two processes occur using two different finite state machines. This concept is illustrated in Fig. 5.12. Synchronization is necessary for the parallel processing of messages. This can be achieved by using flags as indicated in the figure. After the associated data blocks of the second message are processed, the processing of plain-text blocks using the multiplier begins. At this time, the processing of plain-text text blocks of message 1 can be carried out by the second state machine. Note that the tag of the first message needs to be stored for the encryption block to access during processing of message 2.

Application of parallel processing to Deoxys

Deoxys uses a single block cipher for both authentication and encryption. Hence, to apply parallel processing to the Deoxys algorithm, two block ciphers are required. This process is similar to loop unrolling or unfolding technique [20] where the resources are replicated to process more data in the same time frame. Now, the authentication process of message 2 can occur in parallel with the encryption of message 1 blocks.

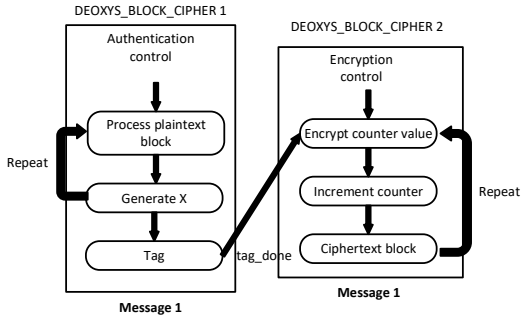


Figure 5.13: Parallel processing of messages in Deoxys.

The two processes can be completely separated since there is no contention of resources between the two processes. The concept of parallel processing in Deoxys is illustrated in Fig. 5.13. The *tag_done* signal which indicates that the tag processing has been completed can itself be used as a flag for the processing of cipher text blocks. Parallely, the authentication state machine can begin the processing of message 2. Note that this results in reduction in time for processing of messages but will also increase the resource consumption because of duplication.

5.5.2 Implementation of S-Box in memory

The discussed algorithms utilize a look-up table in the form of S-Boxes of block ciphers. These look up tables were implemented using a straightforward implementation where the look up tables are synthesized using pure combinational logic elements. However, we note that most modern day device platforms such as FPGAs, microcontrollers *etc.* have some form of memory available on board. This memory can be used to port the look up tables of S-Boxes. This results in reduction of logic elements and utilization of the available memory blocks of the device platform.

5.5.3 Tower field implementation of block cipher

In order to implement the AES algorithm using a compact implementation, the S-Box of AES is replaced using tower field implementations. Basically, the S-Box look up can be considered as an inversion in the Galois field (2^8). By decomposing this field into sub fields, more optimized and compact implementations can be obtained.

These decomposed fields are termed tower fields. Several papers discuss the tower field implementations [90–93]. In this work we adopt two implementations in the form of $GF(((2^2)^2)^2)$ and $GF((2^4)^2)$ decompositions.

5.6 Hardware-Software co-design

Next, we discuss a design style which can serve as a potentially powerful tool in designing for low resource applications in the form of a hardware/software co-design approach. Typically hardware/software co-design is used as a method to increase execution speeds of pure software algorithms by offloading work to dedicated hardware blocks. However, here we look at hardware-software co-design from the perspective of reducing area and optimizing resources. The rationale behind this idea is that while blocks such as AES and Galois Field Multiplication perform efficiently in hardware, the design and optimization of finite state machines on hardware is a challenging task. Implementing this in software simplifies execution while providing for a more flexible and reusable implementation. With modern development platforms typically including a processor, memory and hardware blocks, this approach completely utilizes and optimizes resources in the best possible manner.

5.6.1 Codesign of AES-GCM

A codesign approach is deployed by retaining the AES blocks and Galois Field Multiplier in hardware while implementing the top level state machines in software. The software runs on a processor which could be a specialized processor or reused from one existing on the board. For example when using an Altera FPGA board, we can make use of a 32-bit NIOS II soft processor core by running a C program on it. A top level block diagram of the codesign with the hardware and software boundaries is shown in Fig. 5.14. For accessing data to and from the hardware blocks, we need address decoders which are built as a wrapper around the hardware blocks converting them into what is termed as *custom instructions*. These instructions can then be called as macros in the software program. It is to be noted that data transfer occurs in blocks of 32 bits. However, our design uses 128-bit data requiring a stream of data to be transferred between the hardware and software boundary and synchronized using start and done signals. The

decoders at the hardware/software boundary accept a 4-bit opcode n from the NIOS II processor. Values of n and their corresponding operations are shown in Table 5.2. For each transaction, the NIOS II processor sends out a start signal and waits for a done signal from the decoder block.

5.6.2 Optimization of the co-design

While the co-design approach reduces area and lowers power consumption, it suffers from an increase in latency due to the transfer of data between the processor and AES/Galois Field multiplier blocks. Thus, there is a trade-off between resource consumption *vs* speed of execution. However, optimizations can be made to reduce this latency by minimizing communications between the processor and hardware blocks by making some critical observations. It is seen that the AES block repeatedly uses the secret key while the Galois Field multiplier block needs access to the H value calculated as part of initialization. The two values are marked in Fig. 5.14. Thus, by transferring these values only once and then storing them on the hardware blocks itself, latency per execution of hardware blocks can be reduced.

The approach of using S-Box in memory blocks can also be utilized here since the AES algorithm is still completely present on hardware. This further reduces the area requirement of hardware blocks and utilizes the available memory resources more efficiently. In fact, the data memory accessed by the processor can be shared for the storage of S-Boxes resulting in better resource utilization of the complete system.

5.6.3 Application of co-design techniques to AES-GCM-SIV and Deoxys

The AES-GCM-SIV algorithm matches the AES-GCM algorithm except for the modifications to Galois Filed multiplier blocks (which are necessary to convert them to POLYVAL blocks) and different top-level state machines. Hence, the mapping from hardware to software is the same as that of AES-GCM in this case. All optimizations are equally applicable and the resulting benefits are similar to the AES-GCM algorithm. An alternate approach to co-design of AES-GCM-SIV is to reuse the hardware blocks of AES-GCM and move all changes required by the POLYVAL hashing operations to

Table 5.2: Table of opcodes and their corresponding operations for decoders

Opcode n	**Operations in AES-GCM and AES-GCM-SIV**	**Operations in Deoxys**
0	Write key/H[31:0], key/H[63:32]	Write key[31:0], key[63:32]
1	Write key/H[127:96], key/H[95:64]	Write key[127:96], key[95:64]
2	No operation	Write tweak[31:0], tweak[63:32]
3	No operation	Write tweak[127:96], tweak[95:64]
4	Write data[31:0], data[63:32]	Write data[31:0], data[63:32]
5	Write data[127:96], data[95:64]	Write data[127:96], data[95:64]
6	Perform operation on hardware and wait for done signal	
7	Read result [31:0]	
8	Read result [63:32]	
9	Read result [95:64]	
10	Read result [127:96]	
11-15	No operations	

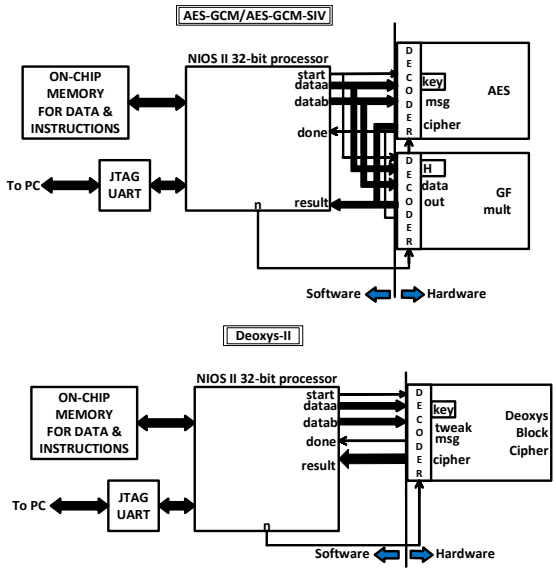


Figure 5.14: Block diagram of a co-design implementation using NIOS II soft processor. The top part of the figure represents co-design on a AES-GCM/AES-GCM-SIV module. The bottom part presents co-design on Deoxys.

software. This results in complete flexibility of reuse where a nonce-misuse resistant and secure AES-GCM-SIV algorithm or the faster AES-GCM algorithm can be chosen based on the application.

The Deoxys algorithm differs by using the Deoxys block cipher for both encryption and authentication. Hence, the Deoxys block cipher is built on hardware similar to the AES block and repeatedly accessed for both encryption and authentication. Because of the requirement of only one hardware block, the overall area of this design will be the lowest indicating that the Deoxys algorithm benefits the most due to co-design approach. Also note that the Deoxys algorithm has an additional tweak input which needs to be provided along with the key and message value. Unlike the key, the tweak input is not constant and hence cannot be stored on the hardware. This requires two additional opcodes (2 and 3) as presented in Table 5.2. Further discussion and comparison between the three algorithms will be presented in the Results section.

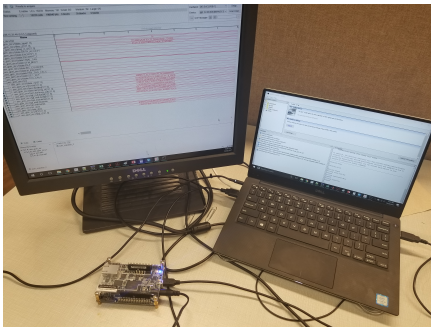


Figure 5.15: Experimental setup of the Atlas SoC board interfaced with a PC using the System Console. The board hosts an Altera Cyclone V FPGA (5CSEMA4U23C6).

5.7 Results

In this section we describe the experimental setup adopted for all implementations and discuss the corresponding results obtained after applying optimizations discussed in Section 5.5. Results are reported both in terms of FPGA and ASIC implementations wherever applicable. This ensures platform obliviousness while making the final comparisons between different algorithms. Note that the described results mainly compare the encryption operation of the AEAD algorithms. Comparison of the decryption or a combination of both is scope for future work.

5.7.1 Experimental setup

All results on the FPGA platform are obtained using Altera's Cyclone V family of FPGA. Specifically, Altera Cyclone 5CSEMA4U23C6 built into an ATLAS SoC board is utilized. Fig. 5.15 illustrates the experimental setup. The Cyclone V family of FPGAs allow for low area, low cost implementations of algorithms. The implementations are written in Verilog Hardware Descriptive Language (HDL) and simulated using the ModelSim tool. Synthesis is performed using the Altera Quartus Prime tool and timing measurements are performed using the TimeQuest timing analyzer. While running synthesis, area optimization is enabled and a target frequency of 50 MHz is used. Power measurements are performed using the PowerPlay power analyzer tool.

For every experiment performed, the following measurements are reported:

- The area consumption is reported both in terms of Look up Tables (LUT), which are the basic combinational elements of FPGAs, as well as register count.

- The output of Power Play power analyzer tool is reported in terms of power in mW. Before running the tool, appropriate inputs in terms of Value Change Dump (VCD) files of the simulation of modules is provided. Details of the generation of power values can be referenced from [8].

- We report the time of operation in terms of cycles per byte. This measure is independent of the FPGA platform and is an important measure of the performance. It is calculated using the following formula:

$$Cycles\_per\_byte = \frac{Cycles\_per\_msg}{Size\_of\_msg\_in\_bytes} \tag{5.4}$$

  The value for the number of cycles is obtained through simulation.

- The throughput is measured using the following equation:

$$Throughput = \frac{Num\_of\_blks \times Blk\_length}{Cycles \times \frac{1}{Max\_freq}} \tag{5.5}$$

  The value of Max_freq is obtained from synthesis. Hence, there is a dependence of the throughput on the FPGA platform. This is different from the cycles per byte measure which is not dependent on the hardware platform used.

- For measurement of energy consumption, the following equation is used:

$$Energy = \frac{Power}{Frequency} \times \frac{Cycles}{Num\_of\_blks \times Blk\_length} \tag{5.6}$$

- The Throughput/Area is measured using the following equation:

$$Throughput/Area = \frac{Throughput\_in\_Mbps}{Area\_in\_LUTs} \tag{5.7}$$

  The Throughput/Area is a good measure which allows comparison between implementations of the different algorithms.

- For measurement of the performance of parallel designs (after application of optimization described in Section 5.5.1), the above measurements are repeated for 100 messages.

All ASIC implementations are synthesized using the *Design Compiler* tool using a 65 nM technology library and target frequency of 50 MHz. The area results are reported in terms of gate equivalent (GE), calculated using the equation:

$$Gate\_Equivalent\ (GE) = \frac{Total\_area\_of\_algorithm}{Area\_of\_NAND\_gate} \tag{5.8}$$

The dynamic power results are reported in mW. This gives a validation for the power numbers reported by FPGA implementations.

5.7.2 Implementation 1: Direct implementation of the algorithms using combinational logic only

The architectural descriptions provided in Section 5.4 are directly mapped to hardware. This results in implementations which are unoptimized but provide a good first comparison between all the algorithms. The results are tabulated in Table 5.3 for a clock speed of 50 MHz. Note that ASIC implementations are also included in the same table for comparison.

From the results we observe that in terms of area, the PRIMATE-APE is the smallest. This can be attributed to the fact that the PRIMATEs are sponge-based designs and hence are expected to be light-weight. The next smallest architecture with respect to area is the Deoxys algorithm. With the modifications of the algorithm as applied to Version v 1.4.1, the area of the algorithm has reduced further. Both versions of Deoxys have the advantage of utilizing just one block cipher for both authentication and encryption processes. This avoids multiplication which is a resource intensive operation. AES-GCM-SIV and POET are slightly more resource consuming in terms of area compared to AES-GCM. The power consumption trends are almost similar to the area consumption trends with PRIMATE and Deoxys having the least FPGA power consumption values.

With respect to performance, AES-GCM has the lowest cycles per byte value. This is because of the inherently parallel nature of this algorithm which is reflected even in the unoptimized implementation of its architecture. With respect to the reported throughput, POET and AES-GCM architectures are the fastest with PRIMATE being the slowest among all. With respect to energy consumption, PRIMATE consumes the

Table 5.3: Area, throughput and power of AES-GCM and other algorithms using direct implementation (Frequency = 50 MHz)

	FPGA results						ASIC results	
Algorithm	**Area**	**Power in mW**	**Cycles per byte**	**Throughput in Mbps**	**Energy in pJ/bit**	**Throughput/Area**	**ASIC area in GE**	**ASIC power in mW**
AES-GCM	LUT : 4087 Registers : 2470	19.52	1.058	417. 16	51.63	0.101	40784	3.317
Deoxys - II (v 1.3)	LUT : 3134 Regs:2084	10.20	1.82	348	46.41	0.111	33422	2.795
Deoxys - II (v 1.4.1)	LUT : 3046 Regs: 2081	9.61	1.82	384.24	43.73	0.126	32616	2.764
AES-GCM-SIV	LUT : 4273 Registers : 2572	17.59	1.46	286.88	61.31	0.067	42995	2.965
POET	LUT : 4824 Registers: 3012	16.14	1.37	476	55.28	0.098	61618	5.031
PRIMATE-APE	LUT: 1699 Registers: 925	3.63	3	194	27.22	0.114	11292	1.276

least energy while AES-GCM-SIV consumes the most energy. The throughput/area indicates that overall, Deoxys has the best ratio. In other words, it has area consumption and performance values in between that of other algorithms making it a good architecture overall. The ASIC implementation results in terms of both area and power are reflective of the FPGA results with PRIMATE and Deoxys consuming the least resources.

Note that both versions of Deoxys have the same cycles per byte value. This is because, the major change in the two versions is with respect to the tweakey key schedule of Deoxys block cipher and initialization method. These affect the maximum frequency reported and hence the throughput of the algorithm and does not affect the cycles per byte value which is a platform independent number.

5.7.3 Implementation 2: Optimization using parallel processing

The optimization described in Section 5.5.1 is applied to both Deoxys and AES-GCM-SIV and the resulting implementation values are reported in Table 5.4. We note that the unoptimized implementation of AES-GCM is based on the standard implementation strategy which makes the implementation inherently parallel. Hence, there is no change in AES-GCM reported values for area, power and timing.

From these results we observe that all algorithms have improved performance benefits after application of the parallel processing technique. The cycles per byte value

Table 5.4: Area, throughput and power of AES-GCM and other algorithms using parallel processing of messages (Frequency = 50 MHz)

	FPGA results						ASIC results	
Algorithm	**Area**	**Power in mW**	**Cycles per byte**	**Throughput in Mbps**	**Energy in pJ/bit**	**Throughput/Area**	**ASIC area in GE**	**ASIC power in mW**
AES-GCM	LUT : 4087 Registers : 2470	19.52	1.058	417. 16	51.63	0.101	40784	3.317
Deoxys - II (v 1.3)	LUT : 4773 Registers : 2773	16.98	1.27	497.63	53.91	0.104	49894	4.361
Deoxys - II (v 1.4.1)	LUT : 5004 Registers : 2859	16.61	1.27	548.03	52.73	0.110	48687	4.332
AES-GCM-SIV	LUT : 4262 Registers : 2668	20.89	1.11	398.55	53.65	0.093	43278	4.170

is significantly reduced and the throughput is significantly increased for all algorithms. However, for parallel processing of Deoxys algorithm, two block ciphers are required. This results in large increase in area consumption for Deoxys compared to AES-GCM-SIV which in fact sees a slight reduction in area consumption. This is because the same blocks of AES-GCM-SIV as used in the serial version are also utilized in the parallel version of the algorithm. The overall throughput/area is high for all algorithms and is comparable to AES-GCM. However, for Deoxys, there is a reduction in the throughput/area number because of the high increase in area even though the throughput also increases significantly, whereas for AES-GCM-SIV, there is improvement in the throughput/area value compared to the serial version. The ASIC area numbers are similar to the area results obtained for FPGA implementation while the ASIC power numbers are more reflective of the results of energy consumption.

5.7.4 Implementation 3: Optimization of the S-Box of the block ciphers using memory blocks

For the implementation optimization described in Section 5.5.2, we consider only Deoxys and AES-GCM-SIV. However, it is to be noted that the optimizations are equally applicable to all the algorithms under study. For Deoxys, the serial implementation is considered since it has a better throughput/area compared to its parallel implementation. However, for AES-GCM-SIV, the parallel implementation of the algorithm is used instead of the serial version for application of the S-Box optimization.

The result of optimization using memory blocks is provided in Table 5.5. The

Table 5.5: Area, Throughput, Power and Memory requirements using optimization of S-Box using memory blocks (Frequency = 50MHz)

Algorithm	Area	Memory	Cycles per byte	Throughput (Mbps)	Power (mW)	Energy pJ/bit	Throughput/Area
AES-GCM	LUT : 3272 Registers : 2413	5 KB	1.058	402.72	21.35	53.01	0.123
Deoxys - II (v 1.3)	LUT : 2482 Registers : 1821	4 KB	1.82	334.64	16.27	74.02	0.135
Deoxys - II (v 1.4.1)	LUT : 2362 Registers : 2081	4 KB	1.82	373.62	13.79	62.74	0.158
AES-GCM-SIV	LUT : 3497 Registers : 2444	5 KB	1.11	308.89	22.18	71.80	0.088

reported results show the memory requirements in kb for storing the S-Box values. We observe from these results that all algorithms map to architectures with lower area and power requirements than their direct or optimized versions. This is because the hardware requirement shifts from using purely combinational logic elements to the memory blocks present on board the FPGA. This occurs at a cost of decrease in the throughput and increase in the energy consumption for almost all algorithms. The modified version of Deoxys reports better improvements compared to the older version since the power consumption is not as high resulting in lower energy consumption values.

5.7.5 Implementation 4: Optimization using tower field implementations

Optimization using tower field implementations are applied to the algorithms which are based on AES. Mainly, optimizations as applied to AES-GCM, AES-GCM-SIV and POET are presented in Fig. 5.6. For the generation of these results we employ two decompositions: $GF((2^4)^2)$ and $GF(((2^2)^2)^2)$ for all the algorithms. Note that the reduction in resources due to compact S-box implementation is not reflected in the implementations on Cyclone V platforms. However, implementations on the ASIC platform reflects this reduction. This can be attributed to the fact that modern FPGA platforms having routing strategies which might result in some optimizations not producing the desired effects. Hence, these optimizations might be more applicable on ASIC platforms or on other FPGA platforms such as Cyclone IV.

Table 5.6: Results of optimization using tower field implementations of AES

	FPGA results						ASIC results	
Algorithm	**Area**	**Power in mW**	**Cycles per byte**	**Throughput in Mbps**	**Energy in pJ/bit**	**Throughput/Area**	**ASIC area in GE**	**ASIC power in mW**
AES-GCM $GF((2^4)^2)$	LUT : 4572 Registers : 2407	15.47	1.058	275.69	40.92	0.060	34827	2.997
AES-GCM $GF(((2^2)^2)^2)$	LUT : 4735 Registers : 2407	17.80	1.058	279.47	47.08	0.059	35307	3.095
AES-GCM-SIV $GF((2^4)^2)$	LUT : 4664 Registers : 2566	19.63	1.11	288.28	54.47	0.062	35138	3.750
AES-GCM-SIV $GF(((2^2)^2)^2)$	LUT : 4808 Registers: 2566	21.81	1.11	252.46	60.52	0.052	35634	3.868
POET $GF((2^4)^2)$	LUT : 5328 Registers : 2878	23.51	1.37	317.95	80.52	0.059	57989	4.569
POET $GF(((2^2)^2)^2)$	LUT : 5424 Registers : 2878	24.26	1.37	301.31	83.09	0.055	58966	4.688

5.7.6 Implementation 5: Optimization using hardware-software co-design approach

The results of implementation of hardware/software co-design approach are reported in Table 5.7. In this table, the area and power of the components of the design built into hardware are shown. The hardware requirements of the NIOS II soft processor are reported separately. Note that this overhead is common for all designs and is a platform specific value. From the implementations, we observe that the area requirements for all designs is reduced significantly. This is because only the hardware modules are now implemented using LUTs and registers. This reduction is obtained at an increased cycle per byte count. Note that there is an almost 10x increase in this value. Also we observe that the component power is quite low compared to the power requirements if the complete module is built using LUTs.

Among the three algorithms, Deoxys benefits the most with a hardware-software codesign approach since the Deoxys algorithm utilizes only one Deoxys hardware block. Hence, the hardware requirements are minimal. Both AES-GCM and AES-GCM-SIV exhibit similar reduction in area and power because of the requirement of both cipher block and multiplier block.

Table 5.7: Results of hardware/software co-design implementation

Algorithm	Area	Cycles per byte	Memory footprint	Component FPGA Power in mW
AES-GCM	LUT: 2562 Registers:1899	19.19	10.507 KB	5.13 (AES), 6.95 (Multiplier)
Deoxys - II (v 1.4.1)	LUT: 1507 Registers: 1080	22.96	7.247 KB	6.00 (Deoxys block cipher)
AES-GCM-SIV	LUT: 2613 Registers: 1666	24.35	9.900 KB	5.43 (AES), 7.34 (Multiplier)
NIOS II processor: 699 LUTs, 584 Registers (32-bit interface) Power consumption = 4.93 mW				

5.8 Analysis of side-channel attacks and Countermeasures

AES-based or block-cipher based schemes are susceptible to side-channel attacks targeting the non-linear S-Box operation or the key schedule/tweak-key schedule. Classical algorithmic and design oriented side-channel attack countermeasures such as masking or randomization schemes [94, 95] for simple (SPA) and differential power analysis (DPA) and fault attacks on AES are applicable for most block-cipher based Authentication Encryption (AE) schemes. Additionally, circuit-level countermeasures such as using dual rail logic style, power supply noise detector or laser attacks shields are useful for all of the AE schemes. Sponge-based AE schemes are more secure against side-channel attacks compared to block-cipher based candidates as they do not have a vulnerable key schedule which is the target of several attacks.

In general, cryptographic implementations can be protected via mechanisms involving frequent re-keying as a countermeasure to DPA attacks. Re-keying achieves this by limiting the number of processed inputs per key for the cryptographic primitive. Each plaintext to be encrypted for the underlying block cipher is provided with a new session key. This session key is derived from a pre-shared master secret and a nonce that is randomly generated on the tag. This inherently prevents DPA on the session key of the block cipher. However, for the session key derivation, a re-keying function that maps the master secret key to the session key and is easy to protect against both SPA and DPA attacks is required. In this context, a new AE candidate, called ISAP [96] has been designed based on the Keyak sponge-based CAESAR candidate, which provides

Table 5.8: Recommendations of candidates based on observed results

Application	**Parameter**	**Candidate**
Resource-constrained	Energy/Power	PRIMATE-APE
High performance	Throughput	POET
Efficient trade-off	Throughput/Area	Deoxys-II
Reusability	Reuse of AES-GCM blocks	AES-GCM-SIV
Small messages	Initialization cycles	Deoxys-II
Software performance	cpb	AES-GCM
Co-design low area requirement	Area/Memory	Deoxys-II

inherent side-channel attack resistance by design. It uses a secure re-keying operation to protect against DPA attacks. In ISAP, several options are proposed for the re-keying function. A masking scheme involving polynomial multiplication of the secret key and the nonce which is strengthened using learning parity with leakage and learning with rounding. Other schemes that are presented are based on the GCM construction that mixes the secret key with the nonce using a tree-based approach and another which also reuses GCM but combines it with a leakage-resistant pseudo-random function.

5.9 Recommendations and Conclusions

In this work we presented the implementation of several nonce-misuse resistant algorithms using both FPGA and ASIC platforms. Based on the results from Section 5.7, we provide recommendations for the candidates most suitable for each application scenario as presented in Table 5.8. Note that all the candidates provide nonce-misuse resistance in some form and hence are already more secure than AES-GCM. Hence, we do not include a recommendation for the candidates in terms of security. Future work in this direction will involve exploring further architecture optimizations of these algorithms.

A preliminary discussion on side-channel analysis was presented in this work. Experimental attacks using side-channels and proposal of countermeasures are topics of further research.

Chapter 6

Low-Energy Architectures of Linear Classifiers

This chapter presents a novel *incremental-precision classification* approach that leads to a reduction in energy consumption of linear classifiers for IoT applications. Features are first input to a low-precision classifier. If the classifier successfully classifies the sample, then the process terminates. Otherwise, the classification performance is incrementally improved by using a classifier of higher precision. This process is repeated until the classification is complete. The argument is that many samples can be classified using the low-precision classifier, leading to a reduction in energy. To achieve incremental-precision, a novel data-path decomposition is proposed to design of *fixed-width* adders and multipliers. These components improve the precision *without* recalculating the outputs, thus reducing energy. Using a linear classification example, it is shown that the proposed incremental-precision based multi-level classifier approach can reduce energy by about 41% while achieving comparable accuracies as that of a full-precision system.

6.1 Introduction

There is a growing trend towards connecting millions of devices to form what is termed as the Internet-of-Things (IoT). Reducing the energy consumption of these IoT devices connected to the network has become critical. Also, there is an increasing need to incorporate machine learning algorithms onto resource-constrained hardware to perform

several tasks. Hence, low-energy solutions for IoT machine learning applications have become important. In this work, we approach classification using bit-width reduction and a multi-level approach. This means that we compute features and perform classification using the lowest possible precision and then improve classification accuracy in a step by step manner based on certain criteria. The argument is that significant amount of data can be classified using the lowest precision components and only the remaining data need to be processed using higher precision components. Thus, overall energy consumption can be reduced.

This work presents an incremental-precision algorithm for classification termed as *multi-level* classification. To process the data in each stage of the algorithm, we need hardware components which start processing in low precision and then increase precision as needed. This necessitates the design of basic components such as adders and multipliers which can perform the required *incremental-precision* operation. The required adders and multipliers are designed using *fixed width* components based on novel data-path decomposition techniques. Using multi-level classification implemented with incremental-precision components, we demonstrate energy reduction in a linear classification system.

6.2 Incremental-precision Addition and Subtraction

Consider the sum of two N bit numbers A and B given by $S = A + B$. If we split A and B into their MSB and LSB parts, we obtain the sum in the following format:

$$S = A_{MSB} + 2^{-p}A_{LSB} + B_{MSB} + 2^{-p}B_{LSB}$$

In this equation, p represents the number of bits of the MSB. The sum can now be rewritten as the following:

$$\begin{aligned} S &= (A_{MSB} + B_{MSB}) + 2^{-p}(A_{LSB} + B_{LSB}) \\ &= S_M + 2^{-p}S_L \end{aligned} \tag{6.1}$$

This is termed as *data-path decomposition* and has been proposed in the context of error-tolerant applications [97].

The hardware architecture of a signed 8-bit incremental-precision adder/subtractor unit decomposed into two 4-bit adders/subtractors is illustrated in Figure 6.1. The top

adder operates on 4 MSB bits of the data and stores it in memory. If a 4-bit output is desired, this output can be read. Instead, if an 8-bit output is desired, the sum of the MSB terms can be read from memory and combined with the output from the lower 4-bit adder/subtractor. The combination equations for both addition and subtraction are also presented in Figure 6.1. Note that the carry information from the lower adder needs to be included in the combination equation. Also since sign bits are considered, appropriate sign extensions are required for subtraction operation. The decomposition can be extended further on the upper or lower adder to create various combinations of incremental precision adders with precision 2-bit/4-bit/6-bit/8-bit. The decomposition of adder/subtraction units and combining the outputs of different precisions leads to an error-free result. However, this is not the case for computing multiplication using incremental-precision, as discussed in next section.

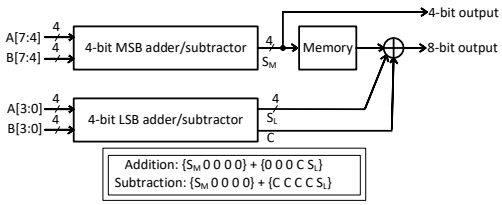


Figure 6.1: Decomposition and subsequent combination operations on 8-bit inputs to create incremental-precision adder/subtraction unit. The 8-bit inputs are decomposed into two 4-bit inputs in this example.

6.3 Incremental-precision Fixed-Width multipliers

Consider the multiplication of two inputs x and y. Data-path decomposition can be applied on the multiplication operation of these two inputs as given by Equation (6.2).

$$P = x \times y = x_{MSB}\ y + 2^{-p}\ x_{LSB}\ y \tag{6.2}$$

x_{LSB} can be further decomposed to produce multiplication in the form of Equation (6.3), where p' is the number of bits in x_{LSB1}.

$$P = x_{MSB}\ y + 2^{-p}\ x_{LSB1}\ y + 2^{-(p+p')}\ x_{LSB2}\ y \tag{6.3}$$

This process can be continued on the LSB bits to produce a decomposed multiplier whose precision improves with every stage. The data flow of the decomposed multiplier is presented in Figure 6.2. To implement the products in this decomposition, fixed-width multipliers are required. Several approaches to implementation of fixed-width multipliers have been presented in the literature. Among them, the approach proposed in [98] is a technique which results in low-error implementations of fixed-width multipliers. This technique is described in detail in the next subsection. The following subsections discuss the modifications required to convert these fixed-width multipliers to *incremental-precision* fixed-width multipliers using certain approximations.

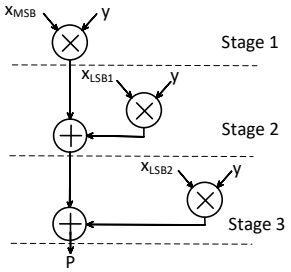


Figure 6.2: Data-path decomposition of multiplier into multiple stages by incrementally increasing the precision of x and maintaining y at a constant precision.

6.3.1 Fixed-width multipliers

Consider multiplication of two inputs x and y of length W bits. The resultant output after multiplication is a value P which is of width $2W-1$ bits. To perform multiplication, the coefficient y is first recoded into y' bits using Booth recoding techniques. This results in less number of partial products that need to be finally summed to generate the terms of the product. The fixed-width multiplier based on Booth recoding is illustrated in Figure 6.3.

For a fixed-width multiplier, we constrain the number of terms in the final output to be of length W bits. If direct truncation is used, the resultant outputs produced will have high error. Hence, appropriate rounding needs to be performed and this idea is illustrated in Figure 6.4. In this figure, the final output is presented as three terms: MP, LP_major and LP_minor which represent the most significant part, the major part of

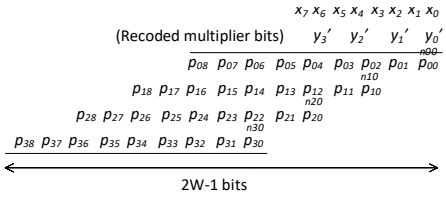


Figure 6.3: Multiplication of two 8-bit inputs by applying Booth recoding on the input y. The resultant output is of length $2W - 1$ bits.

the least significant part, and the minor part of the least significant part, respectively. The MP part needs to be computed precisely. This means that all the partial products required for this part need to be generated. The rest of the output can be approximated using an actual carry and approximate carry value whose details are discussed next. The final carry is termed as *error compensation bias*.

6.3.2 Approximate carry generation

The partial products of LP_major need to be computed precisely. For the terms of the LP_minor, statistical analysis can be used to derive an approximate carry value. Consider Table 6.1 which lists partial products for all possible combinations of recoding (represented by three adjacent bits of y) and combinations of adjacent x inputs. It can be seen that when the recoded value is greater than or equal to 1 (represented by $y'' = 1$), the expected value of partial products is $E[p_{i,j}] = 1/2$. Considering Figure 6.4, we see that the contribution of each y'' bit on the expected value of LP_minor is 2^{-1}. Thus, the total expected value of LP_minor is given by :

$$E[LP\_minor] = 2^{-1}[y''_0 + y''_1 + y''_2] \tag{6.4}$$

The carry signals are generated depending on the values of y''_0 to y''_2. The maximum value of carry in this case occurs when all values of y''_0, y''_1 an y''_2 are equal to 1. This is equal to a value of $E[LP\_minor] = 1.5$ which can be rounded to a value of 2. Note that y''_3 does not contribute to the approximate carry value. For an expected value of 2, the number of carry signals required is 2. Solving for all values of y''_0 to y''_2 using Karnaugh maps, we obtain the required equations for the carry signals as shown in Figure 6.4.

Table 6.1: Table of partial products and the expected values of partial products for every combination of recoded bits y' and input $x_j x_{j-1}$

					Partial products with $x_j x_{j-1}$ ($p_{i,j}$)				**Expected value ($E[p_{i,j}]$)**
y_{2i+1}	y_{2i}	y_{2i-1}	y'	y''	**00**	**01**	**10**	**11**	
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	1	1/2
0	1	0	1	1	0	0	1	1	1/2
0	1	1	2	1	0	1	0	1	1/2
1	0	0	-2	1	1	0	1	0	1/2
1	0	1	-1	1	1	1	0	0	1/2
1	1	0	-1	1	1	1	0	0	1/2
1	1	1	0	0	0	0	0	0	0

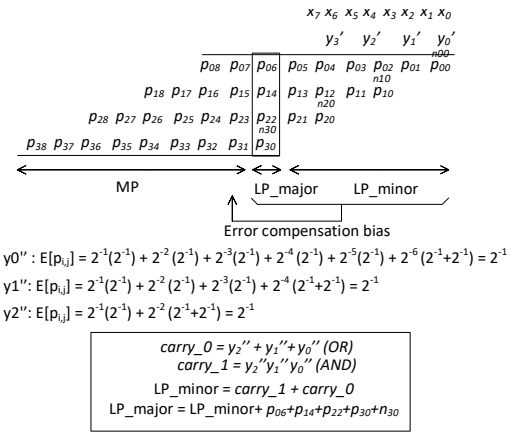


Figure 6.4: Calculation of the error compensation bias terms LP_major and LP_minor based on carry signals.

6.3.3 Two-stage incremental-precision multipliers

From Equation (6.2), we require the computation of two products: $x_{MSB} \times y$ and $x_{LSB} \times y$. Observe that the coefficient y is of the same width while x is split into MSB and LSB parts. Hence, the previous example of 8×8 bit multiplication requires the decomposition of 8-bit operand x into two parts and the multiplication result is approximated. This can be achieved by separating the partial products and redesigning the carry generation circuit as illustrated in Figure 6.5. The top part of the figure

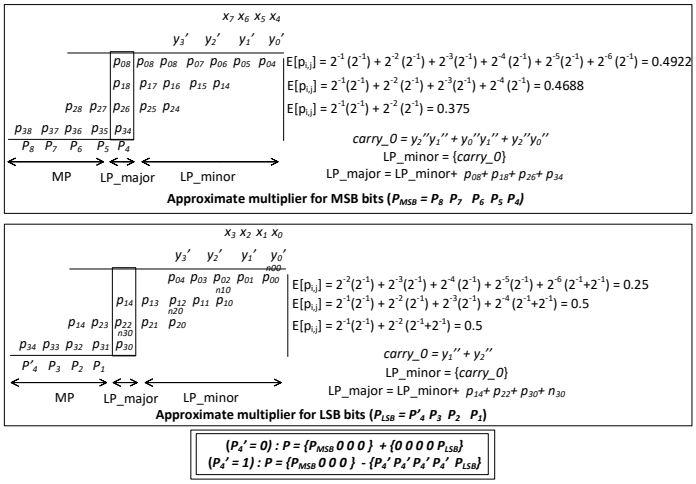


Figure 6.5: Splitting the 8-bit multiplier into two approximate multipliers of 4 bits each.

presents the approximate multiplier required for computation of the MSB bits. Note that the sign bit n_{t0}, where t denotes the row of the partial product, is not present in any of the partial product arrays. However, partial products p_{t8} is required in the calculations. The carry signal is also modified and reduced to just one compared to the 8-bit fixed-width multiplier presented in Figure 6.4.

The bottom part of the figure presents the approximate multiplier required for computation of LSB bits. In this case, the additional partial products (p_{t8}) are not present. However, the sign bit n_{t0} is processed by the bottom part. This requires the calculation of different carry signals. Note again that only one carry signal is required for this multiplier compared to two in the full precision fixed-width multiplier. The two data-paths can now operate *independently* of each other and produce approximate multiplication outputs. The combination of the two products is illustrated in Figure 6.5.

The precision of the 4-bit multiplier operating on the MSB bits is improved by combining the product generated by operating on the LSB bits. The final output obtained through this combination will be an approximation of the full 8-bit precision fixed width output. This is because of the loss of precision during approximation of the products. Approximation is inherently introduced by the fixed-width multipliers as well as the incremental-precision architectures. It may be noted that approximations have been used in other contexts [99–101].

6.3.4 Multi-stage incremental-precision multipliers

The approximate multiplier required to compute the LSB bits can be further split into two as given by Equation (6.3). This idea is illustrated in Figure 6.6. The difference between the two multipliers is that the top multiplier does not account for the sign bit n_{t0} (similar to the multiplier operating on MSB bits). The final output is a combination of the product from MSB bits overlapped with the product from the middle multiplier operating on the first two LSB bits and the last multiplier operating on the last two LSB bits. With each step, precision is improved.

We combine the three multipliers to produce a multi-stage incremental precision multiplier. Note that in the case of splitting an 8-bit multiplier into three stages of 4-bit, 2-bits and 2-bits, stage 2 and stage 3 of 2-bit multiplication can be performed using the same multiplier in a time-multiplexed or folded manner [20]. This idea is illustrated in Figure 6.7. The incremental-precision multiplier structure also requires a memory unit. The computation of intermediate stages are stored in memory.

For example, if only a 4-bit precision output is demanded by the application, the MSB multiplier functions to generate a 4-bit output. The output along with a carry bit are stored in memory. If the application now demands a 6-bit output, the LSB multiplier is functional (with sign bit set to 0) and the memory unit is accessed to retrieve the MSB outputs. The combination of the two outputs is generated as the 6-bit output and also stored in memory along with the carry generated from the second stage. This process continues to provide incrementally higher precision outputs.

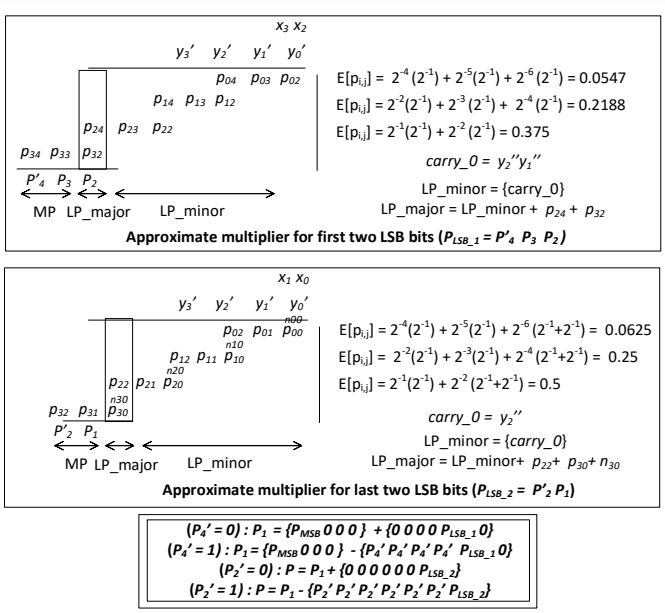


Figure 6.6: Splitting the LSB 4-bit multiplier into two approximate multipliers of 2 bits each.

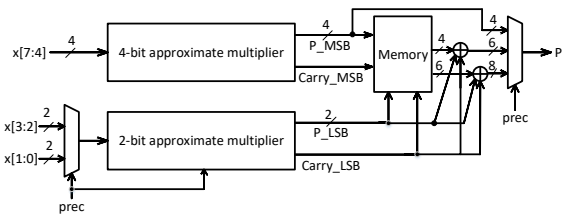


Figure 6.7: Incremental-precision 8-bit multiplier using one 4-bit multiplier and one time-multiplexed 2-bit multiplier. Intermediate outputs are stored in memory.

6.4 Multi-level classification algorithm

The incremental-precision components described in this work can be utilized in a multi-level classification setup where each stage of the classification is based on different precision levels for feature computation as well as classification. In this section, we describe a general multi-level algorithm with a specific example presented in later sections. The basic steps of the algorithm are illustrated in Figure 6.8.

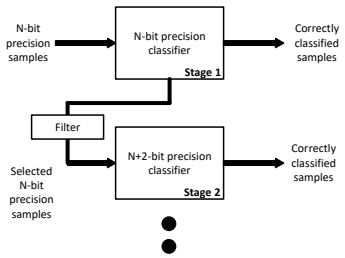


Figure 6.8: Steps of the multi-level classification algorithm with increasing precision classifier at each stage.

Training algorithm:

- Start with a low-precision classifier which operates at a precision of N-bits. Samples which are far away from the classification boundary can be easily classified with this classifier.
- Determine a *threshold* to filter out samples which need to be retrained in higher precision. This threshold could be selected based a distance metric which indicates the distance from the classification boundary.
- Using the determined threshold, select samples to be reclassified in the next higher precision of $N + l$-bits. Here l is the increment in precision between stages.
- This process can be continued till no further improvement in classification accuracy is obtained or for a predetermined number of steps.

Testing algorithm:

- For each test sample, use the first classifier of N-bit precision and obtain a distance metric or probability measure.

- Use the threshold determined in training process to select the samples which need to be reclassified.
- For the test samples which are reselected apply the next $N + l$-bit precision classifier.
- Continue till all stages of classifiers are applied or till no test sample is reselected.

We assume that training occurs offline and the testing/inference algorithm is implemented on hardware. For the first step of testing, a low-precision (N-bit) classifier is used. This implies that all computation can be carried out using the top-level of the decomposed hardware (Figure 6.1 and Figure 6.7). If a sample is reselected to be classified at the next level, the bottom level of the decomposed hardware is used to increment the precision of the classification output. Note that it is not necessary to recompute the first N bits of the classification output. Since most of the samples can be classified using low-precision (N-bit) classifiers, the increment operation is utilized as needed. This leads to significant savings in energy consumption.

6.5 Experimental results

In this section we discuss a binary classification problem based on linear classifiers. Note that even though this is a simple example, the operations involved (namely multiplication with weights and addition of bias) are also fundamental to most machine learning algorithms such as logistic regression, linear Support Vector Machines (SVM), perceptron *etc.* Hence, the techniques in this work can be applied to these algorithms as well after necessary extensions.

6.5.1 Multi-level linear classification

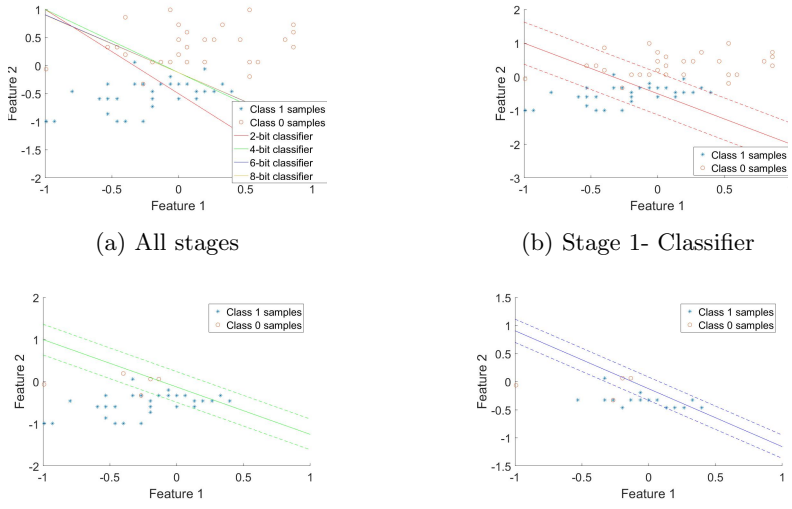
This example considers classification between samples of the Fisher's Iris data set [102]. Two classes of data belonging to two species are picked from this set. There are 50 samples in each category which are randomly divided into training and test data. 60% of the samples form the training set while 40% of the samples are considered for testing. For simplicity, only two features out of the four available features are considered. A

linear classifier attempts to train weights to fit a straight line that separates the data. The classification problem can then be defined as the following:

$$sign(w_1x_1 + w_2x_2 + w_0) > 0 \implies Class = 1 \tag{6.5}$$

In this equation, the features are represented as x_1 and x_2 while the weights are represented as w_0, w_1 and w_2. We observe that while the training process involves calculation of weights, the testing process involves multiplication and addition operations. Hence, we incorporate the incremental-precision components in this classification system.

Consider the multi-level classification algorithm described in the previous section with a starting precision $N = 2$ bits and an increment $l = 2$ bits. The plots of the data with the separating line in different precisions are presented in Figure 6.9. Note that this figure also presents a margin above and below the separating line which represents the *threshold* calculated in the training process. All samples lying beyond this threshold need not be retrained. The calculation of the threshold value is obtained after computing the value $w_1x_1 + w_2x_2 + w_0$ for each sample and setting the threshold equal to the average of the magnitudes of the computed value for all training samples. Note that other techniques for threshold calculations can be used with associated trade-offs.



(a) All stages

(b) Stage 1- Classifier

(c) Stage-2 Classifier

(d) Stage-3 Classifier

Figure 6.9: Classification of samples using multi-level classification algorithm.

Table 6.2: Accuracy of classification for the training and testing process using full-precision single stage and incremental-precision multi-level classifier

Classifier	**Training accuracy**	**Samples trained**	**Testing accuracy**	**Samples tested**
Full-prec (8-bit)	93.33%	100%	95%	100%
Stage 1 (2-bit)	50 %	100%	50%	100%
Stage 2 (4-bit)	93.33%	56.67%	90%	77.50%
Stage 3 (6-bit)	95%	31.67%	95%	20%
Stage 4 (8-bit)	95%	21.67%	95%	12.5%

Table 6.3: Accuracy of the classifier based on implementations of the full-precision and multi-level algorithms

Classifier implemented in hardware	**With original thresholds**		**After recalculation of thresholds**	
	Testing accuracy	**Samples tested**	**Testing accuracy**	**Samples tested**
Full-precision (8-bit)	92.5%	100%	92.5%	100%
Stage-1 (2-bit)	37.5%	100%	37.5%	100%
Stage-2 (4-bit)	72.5%	72.5%	85%	100%
Stage-3 (6-bit)	77.50%	17.5%	92.5%	35%

Table 6.4: Area, Power, Timing and Energy consumption per test sample for the single stage full-precision and multi-level incremental precision algorithms (Numbers in paranthesis indicate values after recalculation of threshold)

Classifier	**Without time-multiplexing**				**With time-multiplexing**			
	Area (um^2)	**Total Power** (mW)	**Total time** (ns)	**Energy per test sample** (pJ)	**Area** (um^2)	**Total Power** (uW)	**Total Time** (ns)	**Energy per test sample** (pJ)
Full-precision (8-bit)	1123	1.480	167	0.155	1123	1.48	167	0.155
Stage-1 (2-bit)	-	0.284 (0.284)	181 (181)	-	-	0.284 (0.284)	181 (181)	-
Stage-2 (4-bit)	-	0.234 (0.323)	150 (207)	-	-	0.249 (0.344)	151 (209)	-
Stage-3 (6-bit)	-	0.071 (0.143)	25 (50)	-	-	0.073 (0.145)	26 (53)	-
Incremental-precision	987	0.589 (0.75)	356 (438)	0.068 (0.086)	867	0.606 (0.773)	358 (443)	0.072 (0.091)

6.5.2 Accuracy of classification

The results of training using a full-precision (8-bit) classifier and the multi-level classification algorithm are presented in Table 6.2. The table also provides the number of samples reselected at each stage. For the multi-level classification algorithm, the class labels for the samples selected at each stage are replaced with new class labels and the classification accuracy for the entire training set is calculated. We observe that the classification accuracy improves with each stage and the number of samples reselected at each step also reduces.

Using the generated classifier models and the calculated thresholds from the training process, we generate the test accuracy at each stage of the algorithm. The test accuracy measures along with the number of reselected samples are also presented in Table 6.2. Stage-4 does not provide any improvement in accuracy and hence the algorithm can be stopped after 3 stages. These results are obtained using an assumption that the components used to implement the testing algorithm provide accurate precision values. In the next section we provide practical implementation results along with the reduction in energy consumption after employing incremental-precision components in hardware.

6.5.3 Test accuracy and reduction in energy using incremental-precision components

The classifier using both single stage and multi-level algorithm is implemented in Verilog Hardware Description Language and synthesized using Design Compiler with a 65 nm technology library. All implementations are clocked at a frequency of 100 MHz and performance is measured in terms of area, power and time required for testing all samples as well as the energy required per sample.

The testing accuracy obtained is reported in Table 6.3. Note that with a full-precision system, an accuracy of 92.5% can be obtained. This is lower than the value obtained through simulation and can be attributed to errors in the multiplication hardware. For the incremental-precision system, we observe that using the original thresholds calculated through simulation, high test accuracies cannot be obtained. This is due to lack of accounting for the additional approximation of the incremental-precision components. After recalculation of the thresholds, the incremental-precision system is

able to achieve the same accuracy as that of the fixed-width full-precision classification system.

Performance measures as well as resource consumption after synthesis are presented in Table 6.4. While the area consumption is not dependent on the number of samples processed at each step, the power consumption and timing requirements are presented based on the number of samples processed. We present the energy consumption per sample after summing the energy requirements for each stage (obtained as a product of the power and timing requirement) and then dividing by the total number of test samples. Implementations both with time-multiplexing of the lower stage and without time-multiplexing are discussed. For each implementation, the values before and after recalculation of thresholds are also presented.

We observe that the energy consumption for the incremental-precision system with the original thresholds is reduced by about 56% compared to the full-precision system. After recalculation of thresholds, the energy reduction is about 45%. With the multiplexed hardware, these values correspond to 53% and 41%, respectively. While the implementation without multiplexing reduces the area of the incremental-precision system by 12%, multiplexing further reduces the area consumption by 22%. The improvements in energy and area consumption are achieved at an expense of increase in the time to process. This is because of the overheads associated with comparison and combination. This timing overhead can be reduced by use of pipelining techniques such that multiple samples can be processed in parallel [20].

6.6 Conclusions

This work presents a novel multi-level classification algorithm and corresponding incremental-precision adders and multipliers that can be used to implement it. Note that the incremental-precision idea is not limited to the classifiers and can be applied to the features as well. Also, the proposed algorithms and architectures are not limited to linear classification and can be used in several machine algorithms with more sophisticated classifiers. The corresponding error-energy trade-off needs to be carefully considered.

Chapter 7

Incremental-Precision based Feature Computation and Multi-Level Classification

This chapter presents a novel technique to reduce energy consumption of a machine learning classifier based on *incremental-precision feature computation and classification*. Specifically, the algorithm starts with features computed using the lowest possible precision. Depending on the classification accuracy, the features of the previous level are combined with features of the incremental-precision to compute the features in higher-precision. This process is continued till a desired accuracy is obtained. A certain *threshold* that allows many samples to be classified using a low-precision classifier can reduce energy consumption, but increases misclassification error. To implement hardware which provides the required updates in precision, an *incremental-precision architecture* based on data-path decomposition is proposed. One novel aspect of this work lies in the design of appropriate thresholds for multi-level classification using training data such that a family of designs can be obtained that enable trade-offs between classification accuracy and energy consumption. Another novel aspect involves the design of hardware architectures based on data-path decomposition which can incrementally increase precision upon demand. Using a seizure detection example, it is shown that the proposed incremental-precision based multi-level classification approach can reduce energy

consumption by 35% while maintaining high sensitivity, or by about 50% at the expense of 15% degradation in sensitivity compared to similar approaches to seizure detection in literature. The reduction in energy is achieved at the expense of small area, timing and memory overheads as multiple classification steps are used instead of a single step.

7.1 Introduction

With ever shrinking devices and embedded platforms, there is an increased demand for energy reduction in applications. Approximate computing has been proposed as an approach to address the energy requirements of modern devices. In this regard, several approximate computing techniques at algorithmic, circuit and synthesis levels have been proposed over the past decade [103]. Reducing word-lengths of components and wires is one of the approaches to achieve energy savings. In this work, we consider approximation as applied to feature extraction unit of classification systems. Feature extraction, which involves extraction of useful information from samples, helps in discriminating the data belonging to different classes. The datapaths of feature extraction consume significant amount of energy. For a complete system-on-chip classifier it is important to focus on the approximation of feature extraction unit which has not been addressed in previous literature.

To apply approximation techniques using word-length reduction we propose a novel *incremental-precision* feature extraction and classification algorithm. The rationale for using incremental-precision is that samples that are far away from the decision boundary can be easily classified using lower precision using a Level-1 classifier. Furthermore, the classifier may require lower precision and may be a simple classifier that requires thresholding. Samples that are not classifiable by the Level-1 classifier can be processed by a Level-2 classifier where the features are computed by incrementally *augmenting* higher precision to the features of earlier precision. This process is repeated as needed to achieve the desired classifier performance.

For the classification of samples from one step to the next, we design a *threshold calculation* unit. This work also proposes the design of *incremental-precision feature computation* which are suitable for the incremental-precision classification paradigm. Specifically, an incremental-precision fast Fourier transform (FFT) design is proposed

in this work. These architectures are based on *data-path decomposition* approaches [97] which are commonly used in error correction schemes but have not been applied in an incremental-precision scenario. *Error models* to understand the errors due to approximation and estimates of power reduction, area reduction/overhead and timing overheads are also presented.

To demonstrate the applicability of incremental precision algorithm and hardware architecture, a complete case study using the seizure detection classification problem is presented. The proposed incremental-precision FFT architecture and its corresponding error and energy models are employed in this design. By studying the *trade-off* between classification accuracy and associated energy reduction, we identify different configurations of values that can be selected to construct the incremental-precision system. An approach to incremental-precision based classification specifically addressing linear classifiers is presented in [11]. The proposed incremental-precision algorithm, incremental-precision architecture and its application to feature extraction are the first in literature and to the best of our knowledge have not been presented before.

7.2 Related work

In this section, we discuss some of the recent literature which address approximate computing algorithms and architectures. Approximate computing has been used in the realm of signal processing applications [99, 100]. With the increasing use of machine learning based techniques, approximate computing has been applied to typical components such as classifiers [104] and synapses of Artificial Neural Networks (ANN) [105]. Bit-width reduction on feature extraction units has been discussed in [106] for the features of an object detection algorithm. An approach to energy reduction by incrementally improving precision of neural networks has been addressed in [107]. However, none of these literature has presented the application of multi-stage bit-width reduction.

The proposed incremental-precision classification approach is similar to the concept of boosting. Boosting is a technique that has been popularly adopted in several machine learning algorithms to design strong classifiers from multiple weak classifiers. The multiple classifiers used in boosting try to improve on the misclassifications of the predecessor [108]. Several applications of the AdaBoost technique to boosting have been

outlined in [109]. In this work, we apply the basic idea of boosting to classification but with the goal of energy reduction in mind. However, there are several differences between the proposed approach and the boosting strategy. These are outlined in Section 7.3.4.

Data-path decomposition has been used to design error-resilient architectures [97]. In this technique, the data-path is divided into most significant bits (MSB) and least significant bits (LSB) components and it is assumed that the two are subject to different types of errors. By combining the two data-paths, error correction can be obtained. The proposed incremental-precision algorithm utilizes a similar approach to decomposing data-paths. The resultant architecture can provide outputs of different precisions by consuming minimal energy.

Incremental-precision architecture is demonstrated using fast Fourier transform (FFT) architectures. Several approaches to both real and complex FFT architectures using ideas of folding, pipelining and parallel processing for both real and complex inputs have been addressed in literature [21, 110]. However, none of these architectures can provide improvements in precision in an incremental manner as required for the proposed low energy algorithm. Approaches to incrementally increasing the length of FFT algorithms has been proposed in [111, 112]. These algorithms improve the SNR by increasing number of stages but do not address lowering precision which is better suited for low-energy multi-level machine learning classifiers. Dynamic Precision has been proposed to adapt the word-length to SNR; however, a single-precision architecture is used in this system [113]. No literature specifically addresses the challenges of FFT data-path decomposition including step wise increments and handling of scaling and other issues.

Seizure detection is used as an example to illustrate the incremental-precision algorithm and its associated architecture. Several techniques to detect seizures have been presented in the literature [114–116]. It has also been shown that using spectral powers as features results in high accuracy for seizure prediction [117]. A low complexity, high accuracy algorithm for seizure prediction has been proposed in [118, 119]. The features used in this algorithm include absolute spectral powers, relative spectral powers and ratios of spectral powers. These features have also been used to detect seizures [120]. However, the energy consumption of these algorithms is still high and can be reduced by using lower precision datapaths. Moreover, support vector machines (SVMs) which

are resource consuming are used in this algorithm. [121] implements seizure detection using Logistic regression (LR) classifiers proving that LR classifiers require the least energy among classifiers such as k-nearest neighbor classifiers, support vector machines with linear and polynomial kernels, naive Bayes and neural networks. Thus, use of LR in seizure detection algorithms can reduce energy as well as complexity.

7.3 Incremental-precision feature extraction and classification system

The proposed incremental-precision algorithm operates on different precision data in an incremental manner starting from the lowest precision. We attempt to classify most of the sample points in low precision before we select samples which need to be processed in the next higher precision. Four important questions need to be addressed:

- How should the samples which need to be reprocessed in the next precision step be selected?
- After how many stages (precision updates), should the algorithm stop?
- What is the value of the lowest precision that the incremental-precision algorithm can start with?
- What are the trade-offs between classification accuracy and energy consumption?

The first two questions are addressed in this section. The last two questions will be discussed using a specific example in Section 7.6.

7.3.1 Training steps

The training steps of the incremental-precision algorithm are described in Fig. 7.1. The first step of the algorithm starts with the lowest precision N. The feature extraction unit operates in N-bit precision in this step. The generated features (F) are passed to a feature selection algorithm to reduce the dimensionality and obtain selected features (SF). A simple classifier is trained and predictions are obtained (denoted as L). We note that along with predicted labels, probability estimates (denoted as P) for each

sample are also required. Probabilities of the classified samples determine the probability of a sample belonging to a particular class. This information will be used for threshold calculation and, hence, the classifier should be able to provide good probability estimates. These probabilities can be obtained by training a logistic regression function on the decision variables of a classifier.

The generated labels (L), probabilities (P) and actual labels (AL) of the training data are then fed to a threshold calculation unit. We will discuss the details of the threshold calculation unit next. However, we observe that the threshold calculation unit generates two thresholds: minimum threshold ($T1$) and maximum threshold ($T2$). The threshold filter then uses P, $T1$ and $T2$ to determine which of the samples are to be reprocessed. This information is provided as indices to the next step of the algorithm. The selected features, classifier model and the threshold values $T1$ and $T2$ are stored and are used for testing of the data. This completes one step of the training process.

For subsequent steps, the unclassified samples from the *threshold filter* are reprocessed in higher precision feature extraction units of precision $N + 2$. Note that other increments in precision can also be used depending on the requirements of the application. The samples are then subjected to the same process of classification and threshold calculation as before. Minimum and maximum thresholds ($T3$ and $T4$, $T5$ and $T6$ etc.) are also calculated for these steps. For the last step of the process, no threshold calculation is performed. The stopping criteria can be pre-determined to be equal to a specified number of steps (which simplifies the algorithm) or when the number of samples reselected is lower than a specified value. Stopping the algorithm when the number of samples reselected is too low prevents overfitting of data.

7.3.2 Threshold calculation

The threshold calculation algorithm is an important step in the incremental-precision algorithm since it determines the number of samples to be selected for reprocessing in higher precision. This affects both the accuracy of the overall classification as well as the energy consumption of the system. The goal of this unit is to try to filter out as many correctly classified samples as possible from the misclassified samples.

A better visualization of the thresholds that need to be calculated is provided in Fig. 7.2. To generate this plot, four arrays are created representing the probabilities of

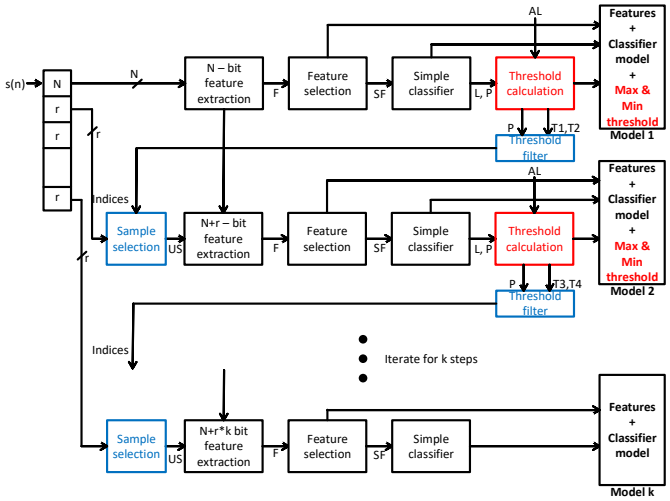


Figure 7.1: Steps of training process of the incremental-precision algorithm using varying precision features.

misclassified samples and correctly classified samples in class 0 and class 1. The arrays are then sorted in ascending order of probabilities. For example, let us consider P to be probability of a sample in class 1. For correctly sampled class 1 points, this value is high. For incorrectly classified class 1 points this value is low. The opposite values hold for class 0 samples. The two thresholds $T1$ (minimum) and $T2$ (maximum) are also marked in this plot. It is to be noted that all sample points having probabilities between minimum and maximum thresholds need to be resampled. By adjusting the value of $T1$ and $T2$, we determine how many points should be reprocessed while ensuring high accuracy values.

Next, we discuss the algorithm for threshold calculation as outlined in Algorithm 1. First, consider the calculation of minimum threshold value ($T1$ for Step 1). Sample points lying above the minimum threshold are to be resampled in the next higher precision. This means that the higher the value of $T1$, less are the number of samples to be reprocessed. However, this also means that more number of blue points (incorrectly classified class 1 samples) are not selected for reprocessing. Thus, to obtain a desirable value, we start with a threshold value of $T1 = 0.1$ (for example) and set the maximum percentage of misclassified class-1 points allowed to a value $M1$. At each iteration, the

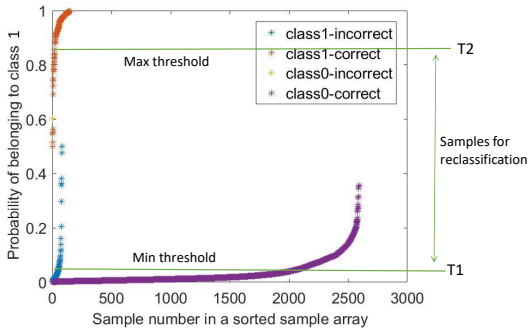


Figure 7.2: Plot of sorted probability estimates of classified data samples for class 0 and class 1 samples. Initial values of thresholds can be set using this data.

threshold is lowered and the percent of misclassified class-1 points are calculated. The algorithm stops when the percent of misclassified class-1 points is lower than $M1$. The final threshold value $T1$ is the output of the algorithm. The initial value of $T1 = 0.1$ is based on the data from Fig. 7.2 and results in faster convergence of the threshold calculation algorithm. In general, the initial value ($init\_val1$) can be selected based on the characteristics of the data being classified. Similar steps can be followed to calculate the maximum threshold value $T2$ using allowed class-0 misclassification error, $M2$, and an initial value of $T2$ equal to $init\_val0$.

Once these thresholds are obtained, the sample probabilities can be used to determine whether processing in the next precision is necessary or not. Thus, the sample is reselected if its probability $P > T1$ and $P < T2$. This is performed in the threshold filtering unit to complete the sample selection process and obtain unclassified samples (US). The number of unclassified samples are then checked to see if they are sufficiently high. For simplicity, we check if $size(US) > 1\% \times size(S)$ where $size(US)$ is the size of reselected sample set and $size(S)$ is the size of the original sample set. If this check passes, the feature extraction and classification continue in the next higher precision. Else, the algorithm is stopped. Note that the stopping criterion of the algorithm can be varied to ensure prevention of overfitting. Similarly the thresholds $T3$ and $T4$ of stage 2 can be calculated using maximum allowed class-1 misclassification error, $M3$, and maximum allowed class-0 misclassification error, $M4$. Thresholds $T5$ and $T6$ of stage 3 can be calculated using $M5$ (maximum allowed class-1 misclassification error)

and $M6$ (maximum allowed class-0 misclassification error) and so on.

7.3.3 Testing steps

The testing algorithm follows the training process of incrementally increasing precision of feature computation. To move from one precision to the next, the threshold values calculated and stored at each step of the training process are used. If the probability estimate of the test sample does not lie in the window formed by the maximum and minimum threshold values, testing is stopped. Else, the sample is reprocessed in the next higher precision. This process is illustrated in Fig. 7.3. We note that even though the training process has an overhead with respect to threshold calculation, this overhead is not reflected in the testing process. During testing, only the stored classification model and calculated threshold values are used for sample selection. We observe that the only overhead in this case is the comparison of the probability with thresholds which can be done with a simple comparator.

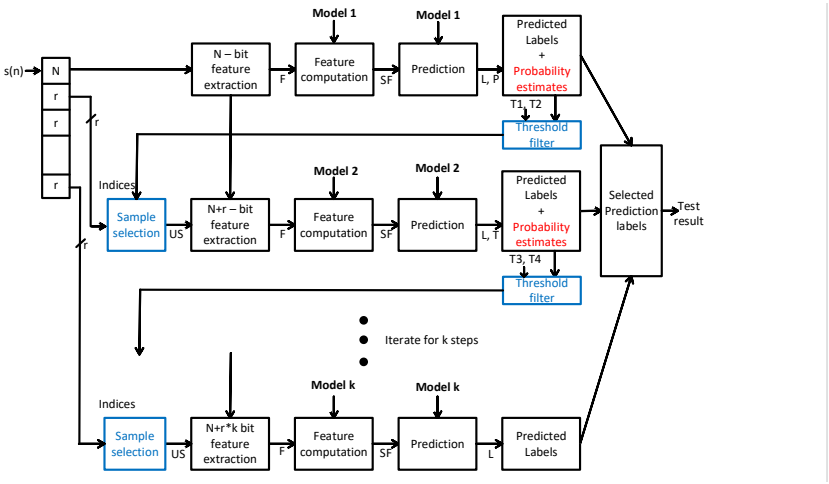


Figure 7.3: Steps of the testing process of incremental-precision algorithm using varying precision features.

7.3.4 Similarities and differences between AdaBoost and incremental precision algorithm

The incremental precision feature extraction and classification algorithm is similar to the AdaBoost algorithm popularly used in machine learning algorithms [108]. The similarities and differences between boosting and our algorithm are presented next.

The AdaBoost algorithm operates by training multiple weak classifiers to the training data in a step-wise manner using weights, eventually resulting in a strong classifier. The weights selected determine the significance of the particular training sample. We summarize the AdaBoost algorithm below:

- Initialize the weights of all samples of the training data to be equal.
- Train a weak classifier and obtain the accuracy in terms of a weighted error (based on both weight of sample and its classification label).
- Update the weights of training samples using the accuracy of previous step.
- Repeat for specified number of steps or till no further improvement in accuracy is obtained.
- Test by applying all weak classifiers on the testing data and use a weighted average (based on the predictions and accuracy of each step) to determine the final prediction label.

Similarly, the target of incremental classification algorithm is to use multiple weak classifiers (due to low precision feature extraction) to obtain a final strong classifier. However, the goal of this system is to minimize energy by minimizing the number of samples that need to be processed in higher precision. Thus the incremental precision algorithm can be summarized as below:

- Begin with a low precision of $N-$bits on all training data.
- Train a simple classifier and obtain the threshold values $T1$ and $T2$.
- Pass data through a threshold filter and update the filtered indices of training data to next higher precision $N+2$. The increment by 2 bits could be varied in

different applications. Note that, unlike AdaBoost, all the training samples are *not* reclassified in this step.

- Repeat for specified number of steps or till number of filtered samples is low. Note that the stopping criteria is based on the size of the training sample set since we are reducing this value in each step.

- Test by applying the classifiers in an incremental manner. Only test samples which are filtered through the thresholds need to be reprocessed. Unlike AdaBoost, all test samples need *not* go through all steps of training. Also, the proposed algorithm does *not* compute a weighted result like in AdaBoost.

‘

7.4 Incremental-Precision Hardware Architectures using Data-Path Decomposition

The incremental-precision classification algorithm is based on the idea of incrementally improving precision of features at each level. This can be achieved by converting the feature extraction process to an incremental-precision system using data-path decomposition of the architecture. Data-path decomposition of adders, multipliers and several other components with respect to error-correction schemes have been addressed in [97]. However, in this work, we approach data-path decomposition such that the complete data-path is decomposed. The goal is to separate the processing of most significant bits and the least significant bits of the system. After computation of the most significant bits of the output, the outputs are stored in memory. For subsequent improvements in accuracy, the output is accessed from memory and combined with the output generated by computation of lower bits. The details of data-path decomposition of a fast Fourier transform (FFT) module is discussed next.

7.4.1 Decomposition of adders and multipliers

The decomposition of addition of two operands x_1 and x_2 can be represented by Equation (7.1). In this equation N represents the number of most significant bits computed

in the first part of the decomposed hardware.

$$\begin{aligned} S &= x_1 + x_2 \\ &= (x_{1M} + 2^{-N} x_{1L}) + (x_{2M} + 2^{-N} x_{2L}) \\ &= (x_{1M} + x_{2M}) + 2^{-N}(x_{1L} + x_{2L}) \end{aligned} \tag{7.1}$$

Similarly, the decomposition of multiplication of two operands x and W is represented by Equation (7.2). In this equation, the value x is decomposed while W is not. W represents coefficients such as filter weights or twiddle factors which are generally stored in memory.

$$\begin{aligned} Y &= x \times W \\ &= (x_M + 2^{-N} x_L) W \\ &= (x_M W) + 2^{-N}(x_L W) \end{aligned} \tag{7.2}$$

To implement these adders and multipliers in fixed-width, appropriate carry generation, approximation of multiplication and carry propagation have to be performed. The details of the decomposition are provided in Appendix A. From the decomposition architectures, it is to be noted that decomposition of addition followed by subsequent combination results in no error. However, decomposition of multiplication followed by combination results in approximation error. We discuss the associated error in Section 7.5.1.

7.4.2 Decomposition of data-path of FFT to form incremental-precision architectures

The idea of incremental-precision architecture is demonstrated using a real FFT architecture as described in [122]. Fig. 7.4 illustrates a 16-point radix-2^2, 2-parallel folded real FFT. This architecture belongs to the class of hybrid architectures where the datapath includes both real and complex data-paths. Folding is a technique to reduction of hardware consumption by time-multiplexing data to reuse components [20]. The paths marked in blue are the complex paths while the rest are real datapaths. Individual components of the data-path which include butterflies (BFI, BFII and BFIV) and Delay-switch-delay (DS) are also defined in Fig. 7.4. The multiplier indicated in the

figure is a complex multiplier which consists of 4 real multipliers. Details of the architecture including the control-path signals are not discussed in this work and the reader is referred to [122]. Radix-2^2 architectures are low-complexity, low area architectures where the multiplier stage is present only in every other stage of the FFT. However, it is to be noted that the discussions below are not limited to these FFT architectures and can be applied in general.

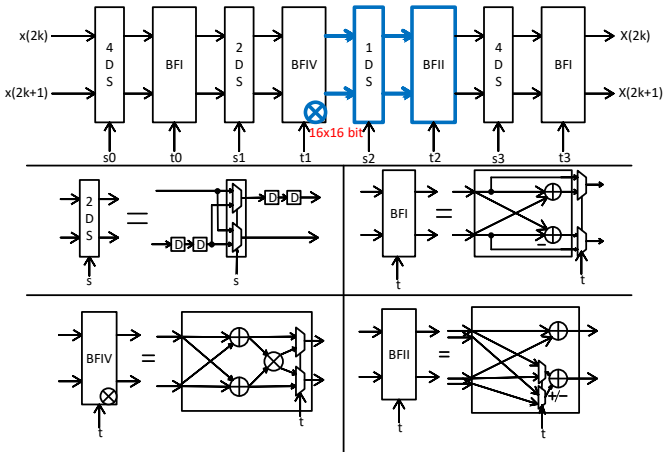


Figure 7.4: 16-point, 2-parallel, folded, radix-2^2 real FFT architecture using a hybrid data-path. The components and paths marked in blue are complex data-paths.

Note that the main components of the FFT architecture include butterfly units and complex multiplication operation. The decomposition of adders and multipliers are applicable to these components. However, in the case of FFT, we *defer* the final combination to the output of the last stage of the FFT. Hence, the butterfly and multiplier operations allow for carry propagation, overflow propagation and any approximations. The details of these decompositions are provided in Appendix B. Note from the decompositions that for correct recombination, the butterfly stages need to increase by one bit at each stage. The decomposed 8-bit/4-bit data-path obtained after the required modifications is illustrated in Fig. 7.5.

In this figure, the top datapath is used for computation of the first 8 bits of the output. After computation, these values are stored in memory. Next, when a 12-bit output is desired, the 8-bit output is accessed from memory and combined with the 4-bit

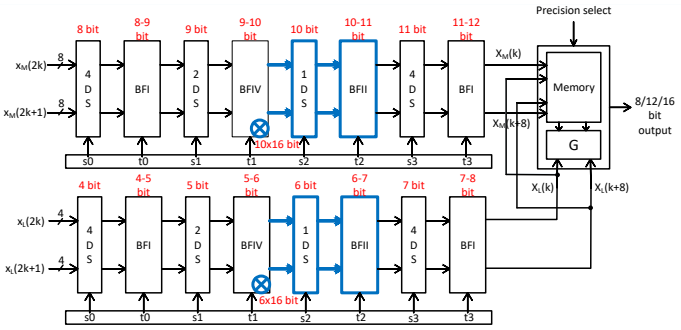


Figure 7.5: Decomposition of 16-point, 2-parallel architecture into upper and lower data-paths with growing bit-widths at each stage. The components and paths marked in blue are complex data-paths. Note that varying precision multipliers are required. The controlpath is common for both the data-paths.

output from the bottom datapath using a combination equation G to produce a 12-bit output. The 12-bit output now replaces the previously stored 8-bit outputs in memory. This process is continued to incrementally improve the precision of the outputs. The final combination equation G for every output $X(K)$ is given by:

$$G : X(k) = X_M(k) + sign\_extend(X_L(k), N) \tag{7.3}$$

where the *sign_extend* function, extends the sign bit of the output of the bottom data-path by the number of bits computed in the top datapath (N).

7.5 Error and Energy requirement analysis

From Section 7.4, the decomposition and subsequent combination of multiplication operation introduces errors in the final output which is in addition to the error introduced due to lowering of precision of inputs. In this section, we model the errors due to these approximations using standard error models. We then contrast the error model with the reduction in resources due to incremental-precision architectures. This provides us with a method to determine if incremental-precision architectures are indeed beneficial to an application or not.

7.5.1 Modeling of error due to input approximation and approximate multipliers

Several studies have been performed to model noise propagated in fixed-point FFT architectures [123, 124]. A simple yet practical approach to model errors in fixed-point implementations of FFT is presented in [125]. Here, we apply similar concepts to the FFT architecture described previously i.e., a 2-parallel, folded, Radix-2^2 architecture. Note that the described error model can be used in general and is not limited to the architecture under discussion.

From [125], the addition and subtraction operations of butterfly units can be modeled as a gain unit of 2 followed by noise source due to rounding. However, if scaling is applied, the model is changed to a gain unit of 1/2 followed by a noise source. For multipliers, recall that each multiplication operation is a complex multiplication consisting of four real multiplication operations. For example if the input is $a + jb$ and the multiplier is $Wr + jWi$, four multiplications are carried out and the sums $aWr - bWi$ and $bWr + aWi$ are generated. Hence, the total error due to multiplication at each output (real and imaginary) is equal to two times the error due to a single multiplication. Using these concepts, for the architecture of a 16-point FFT illustrated in Fig. 7.4, the error model is presented in Fig. 7.6.

In Fig. 7.6, V_{inp} is the variance of the quantization error due to fixed-width inputs. This can be measured using the *Fixed Point Designer ToolBox* of MATLAB and the command $sfi(xinp, prec, prec - 1)$ where $prec$ is the total bit-width, $prec - 1$ is the fractional bit-width and sfi converts the input $xinp$ to a signed fixed point variable. The inputs are assumed to be normalized and represented in the range of -1 to 1 as used in many practical applications. V_{rnd} is the rounding error after butterfly operation. The variance of error is dependent on the bit width of the adders and is proportional to 2^{-2prec} where $prec$ corresponds to the bit-width. For computing the variance of error in multiplication operation (V_{mult}), multipliers are coded in a hardware description language (Verilog) and simulated with 10,000 inputs. The variance of error between these simulations and infinite precision outputs of multiplication is represented as V_{mult}. The total error, V_{total}, which is a combination of V_{inp}, V_{rnd} and V_{mult}, is given by the expression in Fig. 7.6. Note that at each stage, the noise sources are added as linear sources. Due to scaling at each butterfly stage, the error is also scaled.

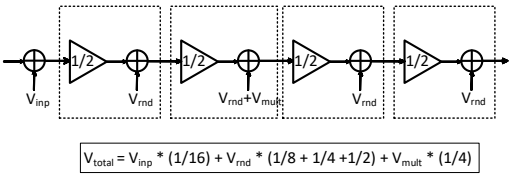


Figure 7.6: Modeling of error in 16-bit datapath of a 16-point radix-2^2 FFT. Note that the butterfly structures are modeled by a gain of 1/2 followed by rounding noise while the multipliers are modeled using an additional noise source at every alternate stage.

While decomposing the datapaths into top and bottom datapaths, we ensure that at each stage the scaled data for top datapath and carry information of bottom datapath are saved. This is achieved by construction of data-paths which grow in precision as described in Section 7.4.2. Hence, the noise introduced at the output due to fixed-width input and rounding at butterfly stage remain the same as that of the error model described in Fig. 7.6. The change in the noise model occurs due to the multiplication operation. This is due to the approximation of multipliers as was discussed in Section 7.4. Hence the noise source V_{mult} of the direct implementation is modified to V_{mult10} and V_{mult6} as illustrated in Fig. 7.7. These variances are also obtained through simulation of RTL of the approximate multipliers of different bit-widths. For simplicity, this figure does not show the errors of butterflies and input error. The error due to approximate multipliers propagates to the outputs as V_{top} for the top datapath and V_{bot} for the bottom datapath. Total error V_{total} is now given by the equation in Fig. 7.7, where the multiplier error is modified to account for the approximate multiplier error and subsequent combination of top and bottom data-paths.

The described error model can be used for any incremental precision decomposition and for any N-point FFT. Fig. 7.8 describes the error obtained for an incremental-precision radix-2^2, 2-parallel, folded 1024-point FFT with various starting precision values, incremental step sizes and stopping precision. The figure also provides the plot of the error obtained with a direct implementation of the architecture at precisions of 4 bits to 16 bits. The starting precision and step increments are limited to powers of 2.

From this error model, we observe that for step increments of 2, the reduction in error stops after a precision of 12-bits is reached. This is because the errors due to

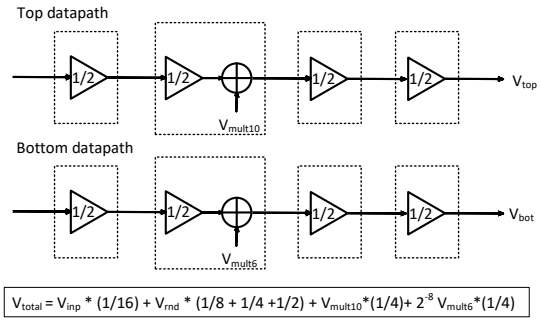


Figure 7.7: Modeling of error in incremental precision architectures due to approximation. The noise due to addition operation V_{rnd} is included in the total error but not shown. The noise sources due to approximate multiplication propagate to the outputs and are added to the total error after combination of data-paths.

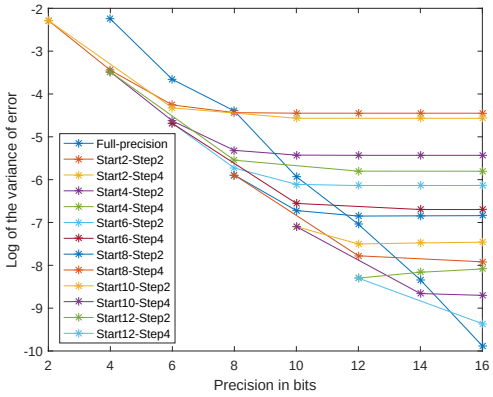


Figure 7.8: Logarithm of the variance of error for different starting precision, step size and final precision for a 1024-point, 2-parallel, radix-2^2 FFT architecture.

approximation exceed the reduction in error due to increase of precision. Hence, only configurations upto 12 bits should be considered. Also observe that while starting at 6, 8 or 10 bits, we are able to reach the precision of 12 bit implementation. However, starting at 4 bits, we are able to only reach the precision of a 10-bit implementation. For step increments of 4 bits, the error reduces upto a precision of 16 bits. However, these increments are better suited for higher starting precisions such as 10 and 12 bits which are able to achieve almost the same accuracy as a direct 16-bit implementation.

7.5.2 Resource consumption and overhead for incremental-precision architectures

The main components of the folded FFT architecture include real multipliers, adders and delay elements. Hence, we consider the resource consumption in terms of these three main resources and understand the reduction of resources *vs.* overhead incurred for various configurations of the incremental-precision architecture. Note that only the major components of the datapath have been considered in our analysis. This is a fair estimate since controlpath of the design is small (less than 1% of the total architecture) and remains the same for both the direct and incremental implementation.

Considering an extension of the architecture as described in Fig. 7.4, we describe the number of resources required for a 16-bit precision 1024-point FFT in Table 7.1. For real adders and multipliers, we list the number of resources as well the size of the components required. From Table 7.1 we observe that for starting precision of 10 bits or higher, the resource consumption is too high and comparable to that of a full 16-bit precision architecture. Hence, these incremental-precision architectures no longer provide sufficient resource reduction compared to accuracy improvement from Fig. 7.8. Hence, we consider starting precisions only upto 8 bits in our design.

To better understand the reduction in resources as well as the overhead incurred, we combine the area and power estimates of the top and bottom parts, and present area-power-timing estimates of different incremental-precision datapaths in Table 7.2. The estimates of the total resources required for a 16-bit precision data-path are also reported. For power estimates, a similar strategy is applied by summing the power consumption of all the components. However, in this case the power consumption of the top and bottom data-paths are reported separately since it is assumed that only

one of the data-paths would be active at any instant of time. The area and power consumption of individual components are obtained through synthesis. For this, Design Compiler synthesis tool is used and all components are synthesized at a frequency of $100MHz$ and a technology library of $65nm$. Note that this is a fairly good estimate of the resource consumption since the most energy hungry components are accounted for in our calculations.

To obtain timing overheads for the design, the critical path is identified and simulated for different precisions. A pipelined FFT architecture is considered in this work with pipeline stages after every multiplier. Hence, it is understood that the critical path lies between two multiplier stages. For a 16-bit data-path, the components between each multiplier are constant and hence the critical path can be considered to be composed of BFII and BFIV of Fig. 7.4. This is indicated in Fig. 7.9. However, due to a growing data-path, the incremental-precision FFT architecture of Fig. 7.5 has a varying critical path. This is also indicated in Fig. 7.9 using a datapath with starting precision of 4-bit as an example. Using the identified paths and simulation results, timing requirements of Table 7.2 can be obtained.

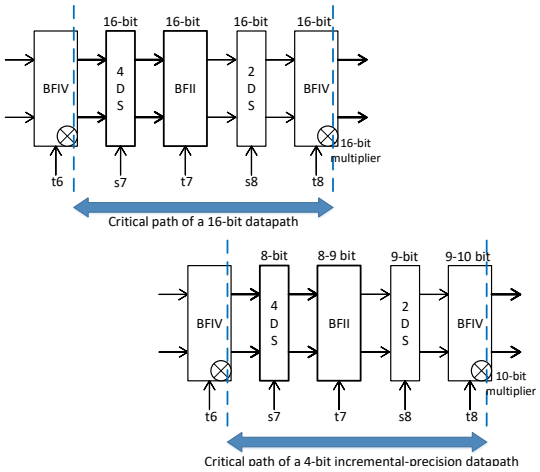


Figure 7.9: Identification of critical path of a 1024-point 16-bit precision FFT and comparison with the critical path of a 1024-point incremental-precision FFT starting at 4 bits. Note that stages 7 and 8 form part of the critical path for incremental- precision architectures due to a growing data-path.

7.6 Case study: Seizure detection using EEG signals

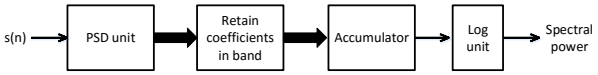
Seizure detection is a binary classification problem which involves classifying between ictal period (when seizure occurs) and interictal period (between seizures) of electroencephalogram (EEG) signals. We label the ictal samples as class 1 and the inter-ictal samples as class 0. For this problem, several distinguishing features can be used for classification. However, it has been observed that use of spectral powers in specific frequency bands of the EEG signals as features is known to demonstrate high separability between the two classes [126]. Implementations of seizure detection and prediction using ratios of spectral power have been presented in [119, 120]. Because of the high accuracy achieved by this system using a low complexity architecture, we use the same algorithm and architecture for our study but with required modifications for incremental-precision systems. For the extraction of spectral powers, an energy hungry power spectral density computation unit is required. Hence, this application is an ideal candidate for implementation using the incremental-precision approach.

7.6.1 Dataset and System description

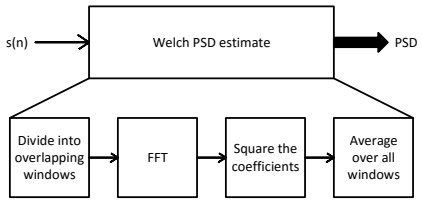
For our analysis, we use the database from UPenn and Mayo Clinic's Seizure Detection Challenge [127]. The feature extraction unit works on EEG data segments labeled as ictal or interictal each of duration $1s$. The sampling frequency is $5000Hz$ which results in 5000 samples to be analyzed for every data segment. 104 features are extracted from each electrode which include absolute spectral power, ratios of spectral powers and relative spectral powers in frequency bands of relevance [120]. Three of the most relevant electrodes are selected using the results published in [120] to generate a total of 312 features which are then ranked according to importance using Classification and Regression Tree (CART). Using these rankings, feature selection is performed to obtain the three best features for each data segment. We replace complex SVM classifiers originally used in [120] with Logistic Regression based classifiers. LR classifiers are known to provide good probability estimates [128] and have low hardware complexity [121]. Since the data is imbalanced, the number of positive samples is low compared to the number of negative samples.

A simplified hardware architecture for extracting spectral power values and power

ratios has been proposed in [119]. The main components of the feature extraction unit are illustrated in Fig. 7.10. The extracted spectral powers are converted to logarithm units which makes it easier to perform addition and subtraction to obtain absolute, relative and ratios of spectral powers. The detailed components required for the power spectral density unit are also shown in this figure. We note that the PSD unit is based on the Welch PSD algorithm which uses overlapped data segments and applies fast Fourier transform (FFT) on each of these segments to extract frequency components of signals [129, 130]. We also observe that FFT is the largest component in the feature extraction unit. By application of the incremental-precision FFT architecture described in Section 7.4.2, we demonstrate reduction in resource consumption.



(a) Feature extraction



(b) Power spectral density estimation

Figure 7.10: Low complexity implementation of feature extraction unit for computation of spectral powers. The Power Spectral density can be implemented using a Welch PSD estimate whose main component is an FFT.

7.6.2 Experimental setup

The errors due to approximation as well as the resource consumption analysis have been discussed in Section 7.5. From Fig. 7.8, we obtain the errors for the various configurations of incremental-precision FFT. The errors are then incorporated into the feature extraction system to generate features of varying precision. Classification is performed using the proposed incremental-precision algorithm for various starting precisions, step

sizes and threshold values. To model resource consumption, the estimates from Table 7.2 are applied to these configurations. For every experiment performed, the data is randomly separated into training and test data with 80% of the data forming the training set and 20% of the data forming the testing set. The experiments performed are based on varying the controllable parameters of the design as follows:

- Even though the algorithm has a stopping criteria dependent on the number of samples reselected, the maximum number of stages allowed can be restricted to a certain value. We experiment with two configurations with maximum values of 3 and 4.

- The maximum allowed misclassification error rates which in turn determine the maximum and minimum threshold values can be varied. When maximum number of stages is 3, the allowed class-0 misclassification error values ($M1$ and $M3$) and the allowed class-1 misclassification error ($M2$ and $M4$) are varied between 0% to 25% in increments of 5%. For simplicity, the values of allowed misclassification error at each stage can be equal, i.e., $M1 = M2$ and $M3 = M4$ etc. This results in 36 configurations. When the maximum number of stages allowed is 4, acceptable misclassification errors ($M1$, $M2$, $M3$, $M4$, $M5$ and $M6$) are varied in the range of 0% to 10% in increments of 5%, resulting in 27 configurations. These configurations are termed as configurations of M values with M ranging from 1 to 36 for a 3-stage algorithm and 1 to 27 for a 4-stage algorithm, respectively.

- The starting precision values are varied between 2 bits to 8 bits and step size is varied between 2 and 4 bits. The starting precision of 10 and 12 bits are not used since the error *vs* energy trade-off for these precisions is not favorable as observed from Fig. 7.8 and Table 7.2.

To measure the improvement in accuracy with respect to varying parameter values, we use both training and test accuracy. Since the data is imbalanced, we use F1-score as a measure for training accuracy. The F1-score is given by Equation (7.4), where $precision = TP/(TP + FP)$ and $recall = TP/(TP + FN)$ and $TP =$ True Positive, $FP =$ False Positive and $FN =$ False Negative.

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{7.4}$$

The test accuracy is better represented in terms of the *Specificity* and *Sensitivity* of the data. While specificity is the true negative rate of the test data, sensitivity corresponds to the true positive rate.

For measuring area, power, timing and energy overheads, the estimates calculated in Table 7.2 are utilized. Assume that the number of samples selected for classification in each stage is stored as t_1, t_2, t_3 etc. Equation (7.5) can then be used to calculate the overall overhead or reduction with respect to a full precision system. In this equation, $Estimate_{top}$, $Estimate_{bot}$ and $Estimate_{full}$ refer to the corresponding area, power or timing estimates of the top, bottom and full-precision data-path, respectively. For energy calculations, $Estimate_{top} = Power_{top} \times Timing_{top}$ where $Power_{top}$ is the power estimate of top datapath and $Timing_{top}$ is the timing estimate of the top datapath. Similarly, $Estimate_{bot} = Power_{bot} \times Timing_{bot}$ where $Power_{bot}$ and $Timing_{bot}$ are the power and timing estimates of the bottom data-path, respectively. Note that the obtained ratio values are calculated per test sample.

$$Ratio = \frac{Estimate_{top}.t1 + Estimate_{bot}.t2 + Estimate_{bot}.t3}{Estimate_{full}.(t1 + t2 + t3)} \tag{7.5}$$

7.7 Results

Based on the proposed incremental-precision classification algorithm, derived error-energy models and the experimental setup described in the previous section, we present results for several configurations. The detailed experimental results are based on Patient 7 of the dataset whose most relevant electrodes have been identified in [120]. The seizure detection example serves as a demonstration of both the multi-level classification algorithm as well as incremental-precision architecture. Overheads of the system are also presented.

7.7.1 Training accuracy

First, we consider the effect of lowering the precision of data as well as the application of LR classifier. The error values obtained for the full-precision system in Fig. 7.8 are used to generate features for this experiment. The results are reported in the plots of Fig. 7.11. From this figure we observe that as the precision of data is increased, both

training and testing accuracy improve. However, it can also be seen that LR classifier is not able to perform well since the maximum sensitivity that can be reached even with the highest precision is about 72%. For high accuracies, SVM classifiers which are energy consuming would be necessary [120] if an incremental-precision algorithm is not employed.

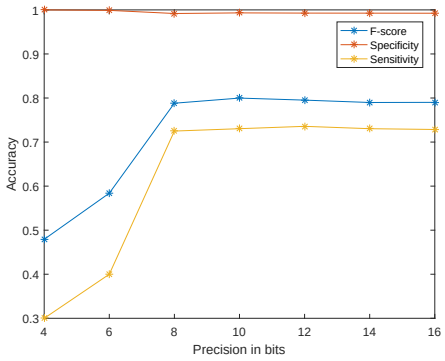


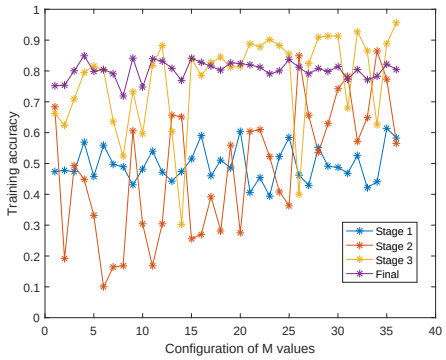
Figure 7.11: Effect of using varying precision features on training accuracy (reported as F1-score) and testing accuracy (reported as Specificity and Sensitivity).

We consider the application of incremental precision algorithm by starting with a precision of 2 bits and incrementing it in step sizes of both 2 and 4. The results for the training accuracy at each increment of the algorithm are presented in Fig. 7.12. From the two subplots we observe that the training accuracy improves significantly from stage 1 to stage 3. Also, increments of step size 4 have higher final accuracies compared to step sizes of 2.

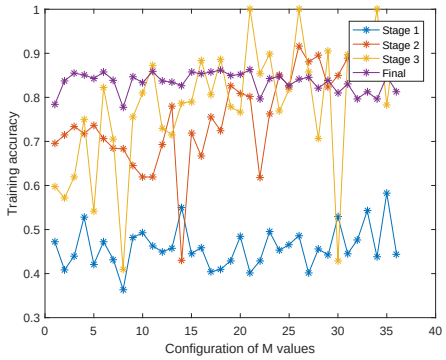
Extending the previous experiment to data-widths ranging from 2 to 8 bits and step sizes of 2 and 4, we obtain the results illustrated in Fig. 7.13. Also, the observations for the algorithm when maximum stages is 4 are presented. It can be observed that the accuracy is highest for the algorithm consisting of 3 stages and a step size of 4.

7.7.2 Testing accuracy and Power reduction

Next, we use the configurations discussed previously to understand the reduction in power *vs.* accuracy achieved. Depending on the number of samples that go through the testing process at each precision, the power consumption is calculated from Table

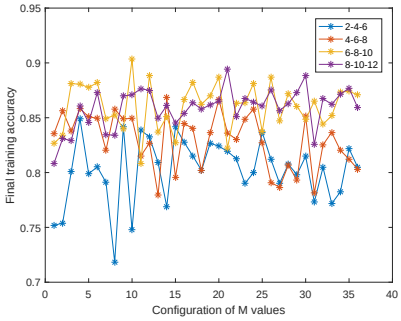


(a) Starting precision = 4 bits, Maximum stages = 3, Step size = 2 bits

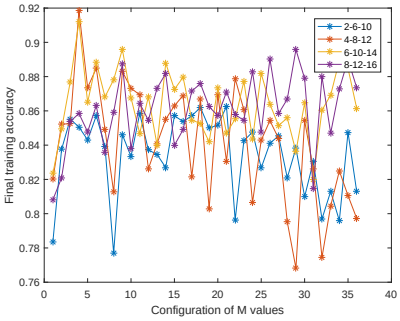


(b) Starting precision = 4 bits, Maximum stages = 3, Step size = 4 bits

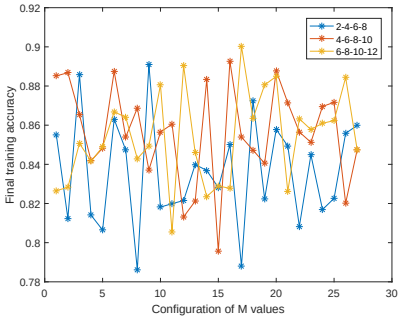
Figure 7.12: Improvement in training accuracy at different stages of the incremental-precision algorithm for various configurations of M values. The x-axis represents different configurations ranging from 1 to 36 with varying maximum allowed misclassification errors $M1$, $M2$, $M3$ and $M4$ (termed as M values).



(a) Starting precision = 2,4,6,8, Step size = 2, Maximum stages = 3



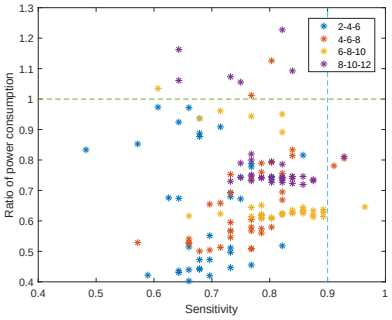
(b) Starting precision = 2,4,6,8, Step size = 4, Maximum stages = 3



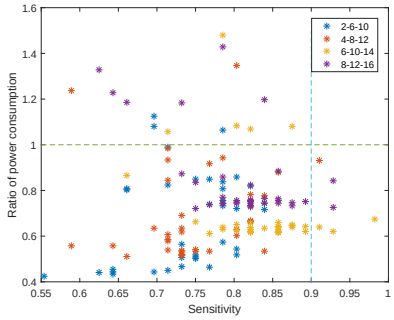
(c) Starting precision = 2,4,6, Step size = 2, Maximum stages = 4

Figure 7.13: Final training accuracy for different starting precision, step size, maximum allowed stages and configurations of M value.

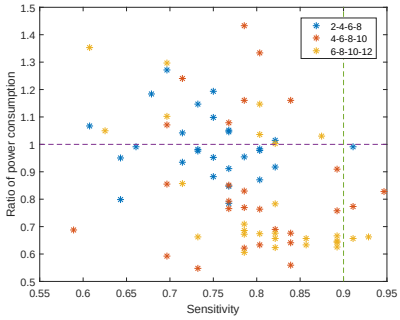
7.2 and Equation (7.5). Since the degradation in precision does not affect specificity, these values are not reported. However, it is ensured that high specificity values are maintained. The plots for trade-off of power reduction (as a ratio) *vs.* sensitivity of testing are presented in Fig. 7.14.



(a) Starting precision = 2,4,6,8, Step size = 2, Maximum stages = 3



(b) Starting precision = 2,4,6,8, Step size = 4, Maximum stages = 3



(c) Starting precision = 2,4,6, Step size = 2, Maximum stages = 4

Figure 7.14: Ratio of power consumption *vs* Sensitivity of test data for different starting precision, step size, maximum stages and configurations of M value. Note that configurations towards the right bottom corner of these plots have high sensitivity and low power ratios.

From these plots, we observe several configurations which provide reduction in power consumption while maintaining high accuracy values (bottom right corner of

the plots). These plots demonstrate the effectiveness of the incremental precision algorithm. However, it is also important to understand the overhead associated with incremental-precision architectures which are studied in the next subsection for selected configurations.

7.7.3 Overhead of selected configurations

Several configurations selected from the power ratio *vs.* sensitivity trade-off plots of previous section are considered and detailed experimental results are presented in Table 7.3. Configuration values including maximum allowed misclassification error rates $M1, M2$, etc., initial and final F1-scores, specificity and sensitivity are also reported. The power, area and timing ratios are reported as percentages of reduction with respect to a full-precision system. Note that there could be a reduction or overhead in area consumption depending on the configuration.

From Table 7.3, we observe that there are several configurations that provide reduction in energy while maintaining high sensitivity values. The configuration with starting precision of 6 bits and step increments of 4 bits results in sensitivity values of 98% while reducing the energy consumption by almost 35%. Also note that a configuration of starting precision of 2 bits and step increment of 2 bits could provide almost 50% reduction in energy while providing sensitivity of 82%. All other reported configurations provide sensitivity values and energy reductions between that of these two configurations. Note that there is a cost associated with the incremental-precision algorithm in terms of the timing overheads. This is because if selected for the consequent stages, the feature extraction process repeats, increasing the computation time for these samples. However, note that for most configurations, the overhead is not too high. This can be attributed to the fact that most of the samples get processed in low precision which occurs faster than high precision computation. The additional memory overhead for this system is not presented in the table. However, it is understood that the 1024-point incremental-precision FFT should have a memory of approximately 2 KB (1024 outputs x 16 bit) to store intermediate values.

Finally, Table 7.4 compares the test accuracy results of our proposed algorithm with the results from [115,116,120]. This table shows that the proposed algorithm maintains high accuracy values similar to other algorithms. For energy consumption comparison,

we compare the results of our work with [119]. This is a fair comparison since the features used in both works are spectral energy ratios. Note that even though other seizure detection algorithms exist which claim lower energy, the goal of our work is to demonstrate applicability of the incremental-precision algorithm and architecture. Our ideas can be applied to other seizure detection or classification setups to result in configurations with lower energy while maintaining high accuracy.

7.8 Conclusions and Future work

We have presented a multi-level classification approach and corresponding incremental-precision datapath architecture methodology. The proposed algorithm and architecture are demonstrated on a seizure detection application which uses spectral powers and ratio of spectral powers as features. The most energy hungry component of spectral power calculation, the FFT architecture, was decomposed using data-path decomposition ideas. The resulting configurations were proven to reduce energy consumption while maintaining high test accuracy values using models which estimate error and resource consumption. The overheads in terms of area increase or timing overhead are also presented. It should be noted that the energy savings are demonstrated to a first-order approximation from synthesis of component parts. Future work will be directed towards synthesis of the complete system. Since the proposed incremental-precision algorithm and incremental-precision architectures are not limited to the demonstrated application, future work will involve applying these ideas to other machine learning applications. Application of data-path decomposition approaches to create incremental-precision architectures for other approximate computing paradigms is also a topic for further research.

7.9 Appendix A

The idea of decomposition of fixed-width addition is presented in Fig. 7.15. From Equation (7.1), the input operands are split such that the most significant part of the operand is composed of N bits. The operands x_1 and x_2 are signed numbers. Hence, padding with zero is necessary to perform addition on the lower bits. Also, a carry bit is

generated out of the lower adder which needs to be propagated to the final combination equation.

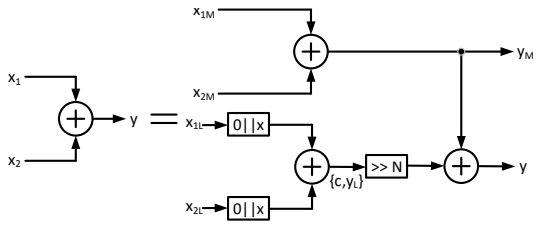


Figure 7.15: Data-path decomposition of fixed-width adder operating on signed numbers x_1 and x_2 and subsequent recombination to produce y. The block $[0||x]$ indicates the operation of appending x with one leading zero.

Several techniques to fixed-width multiplication have been proposed in literature [98, 132]. In this work, we propose tp decompose fixed-width multipliers based on the design presented in [98]. The decomposition of the fixed-width multiplier is illustrated in Fig. 7.16. From Equation (7.2), observe that only the operand x is decomposed. The upper multiplier produces an output of size N bits along with a carry bit C. Similarly, the lower multiplier produces an output of N bits. Depending on the first bit of the lower multiplication output, addition or subtraction has to be performed as part of recombination. Hence, we check the sign and if required invert the output of the lower multiplier. The details of how to decompose the multiplier is discussed next. The carry signals need to be propagated to the final combination equation.

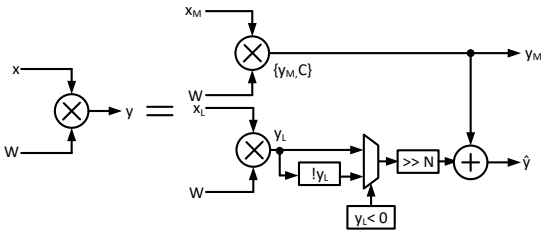


Figure 7.16: Data-path decomposition of fixed-width multiplier operating on x and coefficient W and subsequent recombinaton to produce approximate output $\hat{y}$.

As an example of decomposition of the multiplier, we discuss an 8×8 bit fixed-width

multiplication split into two 4×8 bit multiplications. Fig. 7.17 represents the required partial products and summation for a fixed-width 8×8 bit multiplication on inputs x and y. The multiplication performed here is based on modified Booth's algorithm where the operand y is first recoded to y' whose values can be -2,-1,-0,1 or 2. If y is a value greater 1, a second value y'' is set to 1. This architecture is presented in [98] and the details of calculation of carry signals $carry\_0$ and $carry\_1$ based on y'' can be referenced from the paper.

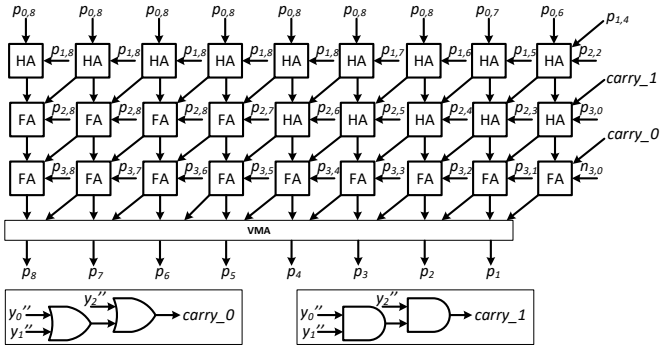


Figure 7.17: Architecture of an 8×8 fixed-width multiplier based on modified Booth multiplication and carry generation.

To decompose the multiplier into two, we split the computation required for the most significant and least significant bits of the multiplier. The decomposed multiplier along with the new carry signals are presented in Fig. 7.18. Note that there is additional approximation involved in the computation of both the most significant and least significant bits of the multiplication operation. Hence, the final output of multiplication $(\hat{y})$ is an approximation of the final $8 - bit$ output. The multiplier introduces an error in the data-path decomposition which needs to be accounted for while calculating the improvement in accuracy from stage to stage.

7.10 Appendix B

In this section, we continue the decomposition to larger components such as butterfly units and complex multipliers. The main challenge in these decompositions is to defer the combination to the end of computation such that the two data-paths are truly

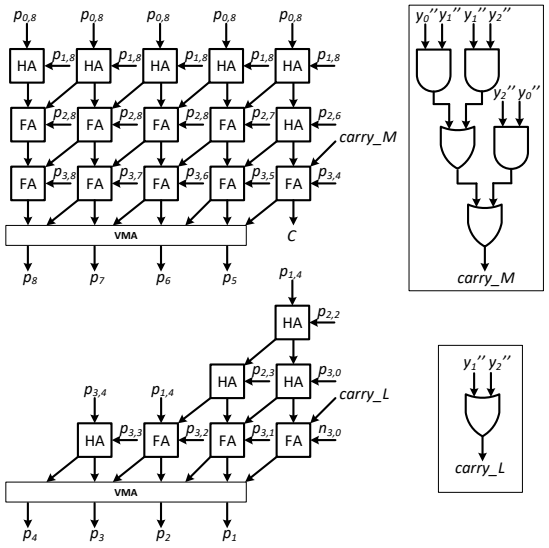


Figure 7.18: Splitting of an 8×8 bit multiplier into two 4×8 bit multipliers by approximation. The carry information from the top multiplier is merged with the bottom multiplier during recombination.

decomposed. The decomposition of the butterfly unit is presented in Fig. 7.19. The butterfly operation involves addition and subtraction followed by a shift to avoid overflow. the butterfly can be decomposed using the concept of adder decomposition as presented in Fig. 7.15. However, there is an additional bit from the upper data path due to shifting (s_1 and s_2) along with the carry information from the lower data path (c_1 and c_2). This means that the top and bottom butterflies need to grow at each stage of the FFT.

The decomposed multiplier presented in Fig. 7.16 is utilized to construct the complex multiplier required for the FFT operation. The component consisting of four multiplication operations and subsequent addition operations is presented in Fig. 7.20. The carry signals from the top multiplier are included in the addition operation.

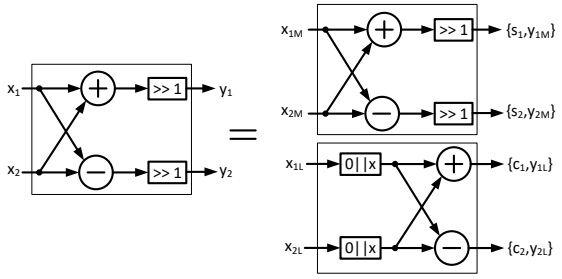


Figure 7.19: Data-path decomposition of the butterfly unit which is composed of addition, subtraction and shift operations. Recombination of outputs is deferred to the last stage.

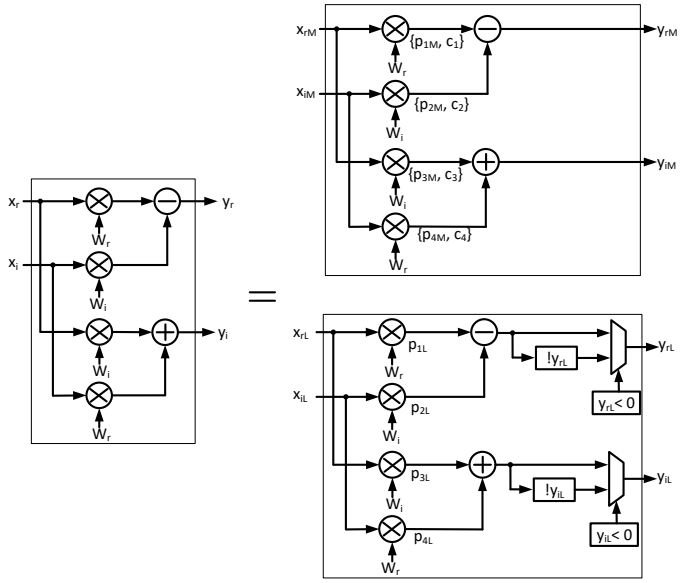


Figure 7.20: Data-path decomposition of the complex multiplication unit composed of four real multiplications, addition and subtraction operations.

Algorithm 1: Algorithm for calculating maximum and minimum threshold values $T1$ and $T2$ of Step 1 of incremental-precision classification algorithm

Output: Final threshold = T1, T2

Input: P (Probability estimates from the classifier),
L (Predicted labels), AL (Actual labels),
M1 (Allowed misclassification error Class-1),
M2 (Allowed misclassification error Class-0)

Initialize: T1 = $init\_val1$; T2 = $init\_val0$;

Compute T1 (Minimum threshold):

while *Class-1 misclassification error* > *M1* **do**
    T1 = T1 - 0.0005;
    for $i = 0;\ i < size(L);\ i = i + 1$ **do**
        if $P < T1$ **then**
            if $AL\ != L$ **then**
                err1 = err1 + 1;
            end
        end
    end
    Class-1 misclassification error = err1 / Total class-1 points;
end

Compute T2 (Maximum threshold):

while *Class-0 misclassification error* > *M2* **do**
    T2 = T2 + 0.0005;
    for $i = 0;\ i < size(L);\ i = i + 1$ **do**
        if $P > T2$ **then**
            if $AL\ != L$ **then**
                err0 = err0 + 1;
            end
        end
    end
    Class-0 misclassification error = err0 / Total class-0 points;
end

Table 7.1: Hardware complexity of a 1024-point FFT in terms of real adders, real multipliers and real delay elements. Cells with two values indicate the resource count and corresponding bit precision in parentheses. The multipliers correspond to 4 stages and the adders correspond to 10 stages.

Datapath	Starting precision	Steps	Real multipliers Count (Precision)	Real Adders Count (Precision)	Real Delay
	16	-	16 (16b)	28 (16b)	27264
Top	**4**	-	4 (6b), 4 (8b), 4(10b), 4(12b)	2(5b), 4(6b), 2(7b), 4(8b), 2(9b), 4(10b), 2(11b), 4(12b), 2(13b), 2(15b)	12876
	6	-	4 (8b), 4 (10b), 4(12b), 4(14b)	2(7b), 4(8b), 2(9b), 4(10b), 2(11b), 4(12b), 2(13b), 4(14b), 2(15b), 2(16b)	16284
	8	-	4 (10b), 4 (12b), 4(14b), 4(16b)	2(9b), 4(10b), 2(11b), 4(12b), 2(13b), 4(14b), 2(15b), 4(16b), 2(16b), 2(16b)	19180
	10	-	4 (12b), 4 (14b), 4(16b), 4(16b)	2(11b), 2(12b), 2(13b), 2(15b), 2(16b), 2(16b), 2(16b), 2(16b), 2(16b), 2(16b)	21552
	12	-	4 (14b), 4 (16b), 4(16b), 4(16b)	2(13b), 2(15b), 2(16b), 2(16b), 2(16b), 2(16b), 2(16b), 2(16b), 2(16b), 2(16b)	23872
Bottom	-	2	4 (4b), 4 (6b), 4(8b), 4(10b)	2(3b), 4(4b), 2(5b), 4(6b), 2(7b), 4(8b), 2(9b), 4(10b), 2(11b), 2(12b)	9468
	-	4	4 (6b), 4 (8b), 4(10b), 4(12b)	2(5b), 4(6b), 2(7b), 4(8b), 2(9b), 4(10b), 2(11b), 4(12b), 2(13b), 2(14b)	12876

Table 7.2: Estimates of area, power consumption and timing of critical path of the 1024-point incremental-precision architecture compared to a 16-bit precision architecture. The resource consumption values and performance numbers are obtained using synthesis of components with Design Compiler, 65nM technology library and a clock frequency of 100MHz

Starting precision	**Steps**	**Area in um^2**	**Power in mW (top)**	**Power in mW (bottom)**	**Timing in ns (top)**	**Timing in ns (bottom)**
16	-	337450	29.0134	-	9.85	-
2	2	223790	9.8392	9.8392	9.421	9.421
	4	264160	9.8392	13.408	9.421	9.469
4	2	264160	13.408	9.8392	9.469	9.421
	4	304540	13.408	13.408	9.469	9.469
6	2	305140	17.0053	9.8392	9.538	9.421
	4	345520	17.0053	13.408	9.538	9.469
8	2	346160	20.2719	9.8392	9.826	9.421
	4	386540	20.2719	13.408	9.826	9.469

Table 7.3: Performance of selected configurations with respect to final training accuracy, specificity, sensitivity, power reduction, area reduction, timing overheads and energy consumption

Precision increments	M1/M2 (Step 1)	M3/M4 (Step 2)	M5/M6 (Step 3)	F1-score (Step 1)	F1-score (Final)	SP	SS	Power reduction	Area reduction	Timing overhead	Energy requirement
16-0-0	-	-	-	-	0.790	0.992	0.728	0%	0%	0%	100%
6-10-14	5%	10%	-	0.786	0.874	0.988	0.982	32.59%	- 2.39%	15.13%	65.21%
6-8-10	5%	10%	-	0.789	0.855	0.985	0.964	35.41%	9.57%	13.68%	62.47%
4-6-8-10	0%	10%	10%	0.591	0.906	0.992	0.946	17.13%	21.72%	99.52%	79.49%
6-8-10	15%	0%	-	0.771	0.874	0.992	0.893	36.28%	9.57%	11.23%	61.64%
8-10-12	10%	20%	-	0.791	0.860	0.989	0.875	26.76%	- 2.58%	9.27%	72.93%
4-8-12	15%	25%	-	0.634	0.807	0.975	0.839	46.57%	9.75%	11.15%	51.37%
6-10-14	15%	25%	-	0.794	0.862	0.991	0.929	37.84%	- 2.39%	4.21%	60.16%
6-10-14	20%	5%	-	0.779	0.859	0.994	0.911	36.01%	- 2.39%	8.03%	61.93%
8-12-16	10%	25%	-	0.839	0.887	0.995	0.928	27.40%	- 14.54%	5.49%	72.35%
6-8-10-12	10%	0%	0%	0.771	0.879	0.992	0.928	33.68%	9.57%	18.57%	64.13%
6-8-10-12	10%	10%	0%	0.783	0.863	0.983	0.893	37.6%	9.57%	7.70%	60.44%
2-4-6	20%	0%	-	0.559	0.848	0.995	0.821	48.26%	33.68%	45.91%	49.48%

Table 7.4: Comparison of testing accuracy and energy consumption of proposed incremental-precision system applied to seizure detection with similar approaches from literature

Comparison of test accuracy		
Algorithm	**Specificity**	**Sensitivity**
Ratio of spectral power + SVM classifier [120]	99.90	100
Ratio of spectral power + LR classifier	99.24	72.86
[115]	99.19	91.29
[116]	94.89	91.72
Proposed approach	98.88	98.20
Comparison of energy consumption		
Algorithm	**Energy of feature selection**	**Energy of classifier**
Ratio of spectral power + SVM classifier [120]	226.26 nJ (Full-precision FFT)	∼60 uJ [131]
Proposed approach	∼ 147.07 nJ (65% of full-precision FFT)	∼0.15 uJ [131]

Chapter 8

Conclusion and Future Work

This thesis presented architectures targeted at three major application areas. For security in manufacturing of semiconductor circuits, mode based obfuscation which is a fixed method of obfuscation was introduced. The shortcomings of such a technique were presented and a novel method of dynamic obfuscation was proposed. The dynamic obfuscation guarantees better security using smaller key sizes and increases the lower bound of security from 1 to $K2^K$. A detailed algorithm for dynamic obfuscation which can be adopted by CAD tools was also presented.

For resiliency in communication, authenticated encryption algorithms and architectures where discussed in detail. The importance of nonce-misuse resistance property in authenticated encryption algorithm especially when implemented on a low resource device was highlighted. Several algorithms such as the Deoxys, AES-GCM-SIV, POET and PRIMATE where discussed in detail, implemented on hardware, optimized and compared with the AES-GCM standard.

With respect to low-energy classification, incremental-precision algorithm for classification and architectures based on adders and multipliers was presented. The applicability of this algorithm to reduce energy consumption in linear classification as well as in applications of seizure detection were demonstrated in detail. It was shown that the algorithm is most suitable for applications where most of the data can be classified using low precision and that there is improvement in energy reduction as more data is processed over time.

Future work in these areas of application were discussed at the end of each chapter.

Here, we point out some other directions of future work. Specifically, for hardware obfuscation, testing of the encrypted integrated circuit can be a challenge that needs to be addressed. The technique can also be made stronger by deriving different keys for different chips similar to a popular idea of IC metering. A thorough analysis of side-channel attacks and extension of the idea to split manufacturing are also scope for future work.

For the architecture of authenticated encryption algorithms, it is highly important to study side-channel attacks. Specifically power, timing, electromagnetic and cache based attacks launched using specialized FPGA boards such as the Sasebo board is part of future work.

Finally, with respect to the incremental-precision architectures, the decomposition of multipliers using data-path decomposition needs to be improved for better accuracy. The application of the incremental-precision technique to architectures such as FIR and IIR filters is also scope for future work. Extending the incremental-precision algorithm to deep neural networks and incorporating training on hardware would also be of high interest to the IoT research community.

References

[1] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review*, 45(5):37–42, 2015.

[2] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[3] Sandhya Koteshwara, Chris H Kim, and Keshab K Parhi. Mode-based Obfuscation using Control-Flow Modifications. In *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems*, pages 19–24. ACM, 2016.

[4] Sandhya Koteshwara, Chris H Kim, and Keshab K Parhi. Hierarchical functional obfuscation of integratec circuits using a mode-based approach. In *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*, pages 1–4. IEEE, 2017.

[5] Sandhya Koteshwara, Chris H Kim, and Keshab K Parhi. Key-based dynamic functional obfuscation of integrated circuits using sequentially triggered mode-based design. *IEEE Transactions on Information Forensics and Security*, 13(1):79–93, 2018.

[6] Sandhya Koteshwara, Chris Kim, and Keshab K Parhi. Functional Encryption of Integrated Circuits by Key-Based Dynamic Hybrid Obfuscation. In *Proc. 2017 Asilomar Conference on Signals, Systems and Computers*, pages 484–488. IEEE, 2017.

[7] Sandhya Koteshwara and Amitabh Das. Comparative study of Authenticated Encryption targeting lightweight IoT applications. *IEEE Design & Test*, 2017.

[8] Sandhya Koteshwara, Amitabh Das, and Keshab K Parhi. FPGA implementation and comparison of AES-GCM and Deoxys authenticated encryption schemes. In *IEEE International Symposium on Circuits and Systems (ISCAS), 2017*, pages 1–4. IEEE, 2017.

[9] Sandhya Koteshwara, Amitabh Das, and Keshab K Parhi. Performance Comparison of AES-GCM-SIV and AES-GCM Algorithms for Authenticated Encryption on FPGA Platforms. In *Proc. 2017 Asilomar Conference on Signals, Systems and Computers*, pages 1331–1336. IEEE, 2017.

[10] Sandhya Koteshwara, Amitabh Das, and Keshab K Parhi. Architecture Optimization and Performance Comparison of Nonce-Misuse Resistant Authenticated Encryption Algorithms (Submitted). *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

[11] Sandhya Koteshwara and Keshab K Parhi. Low-energy architectures of linear classifiers for iot applications using incremental precision and multi-level classification. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, pages 291–296. ACM, 2018.

[12] Sandhya Koteshwara and Keshab K Parhi. Incremental-precision based feature computation and multi-level classification for low-energy internet-of-things. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2018.

[13] Masoud Rostami, Farinaz Koushanfar, Jeyavijayan Rajendran, and Ramesh Karri. Hardware security: Threat models and metrics. In *Proceedings of the International Conference on Computer-Aided Design*, pages 819–823, 2013.

[14] Maciej Brzozowski and Vyacheslav N Yarmolik. Obfuscation as intellectual rights protection in vhdl language. In *Proceedings of the 6th International Conference on Computer Information Systems and Industrial Management Applications (CISIM)*, pages 337–340, 2007.

[15] Rajat Subhra Chakraborty and Swarup Bhunia. RTL hardware IP protection using key-based control and data flow obfuscation. In *Proceedings of the 23rd International Conference on VLSI Design*, pages 405–410, 2010.

[16] Jarrod A Roy, Farinaz Koushanfar, and Igor L Markov. EPIC: Ending piracy of integrated circuits. In *Proceedings of the conference on Design, automation and test in Europe*, pages 1069–1074. ACM, 2008.

[17] Rajat Subhra Chakraborty and Swarup Bhunia. HARPOON: an obfuscation-based SoC design methodology for hardware protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1493–1502, 2009.

[18] Yingjie Lao and Keshab K Parhi. Obfuscating DSP circuits via high-level transformations. *IEEE Transactions on VLSI Systems*, pages 819–830, May 2015.

[19] Goutham NC Shanmugam, Yingjie Lao, and Keshab K Parhi. An obfuscated radix-2 real FFT architecture. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1056–1060. IEEE, 2015.

[20] Keshab K Parhi. *VLSI digital signal processing systems: design and implementation*. Wiley, New York, 1999.

[21] Manohar Ayinala, Michael Brown, and Keshab K Parhi. Pipelined parallel FFT architectures via folding transformation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(6):1068–1081, 2012.

[22] Jeyavijayan Rajendran, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Security analysis of logic obfuscation. In *Proceedings of the 49th Annual Design Automation Conference*, pages 83–89, 2012.

[23] Masoud Rostami, Farinaz Koushanfar, and Ramesh Karri. A primer on hardware security: Models, methods, and metrics. *Proceedings of the IEEE*, 102(8):1283–1295, 2014.

[24] John Villasenor and Mohammed Tehranipoor. Chop shop electronics. *IEEE Spectrum*, 50(10):41–45, 2013.

[25] Keshab K Parhi. Verifying equivalence of digital signal processing circuits. In *Signals, Systems and Computers (ASILOMAR), 2012 Conference Record of the Forty Sixth Asilomar Conference on*, pages 99–103. IEEE, 2012.

[26] Rajat Subhra Chakraborty and Swarup Bhunia. Hardware protection and authentication through netlist level obfuscation. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 674–677. IEEE Press, 2008.

[27] Jeyavijayan Rajendran, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Logic encryption: A fault analysis perspective. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 953–958. EDA Consortium, 2012.

[28] Jiliang Zhang. A practical logic obfuscation technique for hardware security. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(3):1193–1197, 2016.

[29] Sophie Dupuis, Papa-Sidi Ba, Giorgio Di Natale, Marie-Lise Flottes, and Bruno Rouzeyre. A novel hardware logic encryption technique for thwarting illegal overproduction and Hardware Trojans. In *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, pages 49–54. IEEE, 2014.

[30] Pramod Subramanyan, Nestan Tsiskaridze, Kanika Pasricha, Dillon Reisman, Adriana Susnea, and Sharad Malik. Reverse engineering digital circuits using functional analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1277–1280. EDA Consortium, 2013.

[31] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2015*, pages 137–143. IEEE, 2015.

[32] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. SARLock: SAT attack resistant logic locking. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2016*, pages 236–241. IEEE, 2016.

[33] Farinaz Koushanfar. Provably secure active IC metering techniques for piracy avoidance and digital rights management. *IEEE Transactions on Information Forensics and Security*, 7(1):51–63, 2012.

[34] Jiliang Zhang, Yaping Lin, Yongqiang Lyu, and Gang Qu. A PUF-FSM binding scheme for FPGA IP protection and pay-per-device licensing. *IEEE Transactions on Information Forensics and Security*, 10(6):1137–1150, 2015.

[35] Yousra Alkabani, Farinaz Koushanfar, and Miodrag Potkonjak. Remote activation of ICs for piracy prevention and digital right management. In *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, pages 674–677. IEEE Press, 2007.

[36] Kyle Juretus and Ioannis Savidis. Reducing logic encryption overhead through gate level key insertion. In *IEEE International Symposium on Circuits and Systems (ISCAS), 2016*, pages 1714–1717. IEEE, 2016.

[37] Yingjie Lao and Keshab K Parhi. Obfuscating DSP circuits via high-level transformations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(5):819–830, 2015.

[38] Aijiao Cui, Yanhui Luo, and Chip-Hong Chang. Static and Dynamic Obfuscations of Scan Data Against Scan-Based Side-Channel Attacks. *IEEE Transactions on Information Forensics and Security*, 12(2):363–376, 2017.

[39] Seetharam Narasimhan, Rajat Subhra Chakraborty, and Swarup Bhunia. Hardware IP protection during evaluation using embedded sequential trojan. *IEEE Design & Test of Computers*, 29(3):70–79, 2012.

[40] Swarup Bhunia, Michael S Hsiao, Mainak Banga, and Seetharam Narasimhan. Hardware Trojan attacks: threat analysis and countermeasures. *Proceedings of the IEEE*, 102(8):1229–1247, 2014.

[41] Rajat Subhra Chakraborty, Seetharam Narasimhan, and Swarup Bhunia. Hardware Trojan: Threats and emerging solutions. In *High Level Design Validation and Test Workshop, 2009. HLDVT 2009. IEEE International*, pages 166–171. IEEE, 2009.

[42] Yang Xie and Ankur Srivastava. Mitigating sat attack on logic locking. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 127–146. Springer, 2016.

[43] ISCAS High-Level Models. `http://web.eecs.umich.edu/~jhayes/iscas.restore/benchmark.html`.

[44] Ethernet MAC 10/100 Mbps. `https://opencores.org/project,ethmac`.

[45] Sandhya Koteshwara, Chris H Kim, and Keshab K Parhi. Hierarchical functional obfuscation of integrated circuits using a mode-based approach. In *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*. IEEE, 2017.

[46] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 531–545. Springer, 2000.

[47] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM conference on Computer and Communications Security*, pages 98–107. ACM, 2002.

[48] David McGrew and John Viega. The Galois/counter mode of operation (GCM). *Submission to NIST. http://csrc.nist.gov/CryptoToolkit/modes/proposed modes/gcm/gcm-spec.pdf*, 2004.

[49] Morris J Dworkin. *Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality*. 2007.

[50] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX mode of operation. In *International Workshop on Fast Software Encryption*, pages 389–407. Springer, 2004.

[51] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transactions on Information and System Security (TISSEC)*, 6(3):365–403, 2003.

[52] Gang Zhou, Harald Michalik, and Laszlo Hinsenkamp. Efficient and high-throughput implementations of AES-GCM on FPGAs. In *International Conference on Field-Programmable Technology, 2007. ICFPT 2007.*, pages 185–192. IEEE, 2007.

[53] Mehran Mozaffari-Kermani and Arash Reyhani-Masoleh. Efficient and high-performance parallel hardware architectures for the AES-GCM. *IEEE Transactions on Computers,*, 61(8):1165–1178, 2012.

[54] Markku-Juhani Olavi Saarinen. Cycling attacks on GCM, GHASH and other polynomial MACs and hashes. In *Fast Software Encryption*, pages 216–225. Springer, 2012.

[55] Hanno Böck, Aaron Zauner, Sean Devlin, Juraj Somorovsky, and Philipp Jovanovic. Nonce-Disrespecting Adversaries: Practical Forgery Attacks on GCM in TLS. *USENIX WOOT*, 2016.

[56] `http://competitions.cr.yp.to/caesar-call.html`, 2014.

[57] Farzaneh Abed, Christian Forler, and Stefan Lucks. General overview of the authenticated schemes for the first round of the caesar competition. Technical report, Cryptology ePrint Archive: Report 2014/792.[2] CAESAR submissions, second-round candidates. Available: http://competitions. cr. yp. to/caesar-submissions. html.

[58] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-based encryption, authentication and authenticated encryption. *Directions in Authenticated Ciphers*, 2012.

[59] Chanathip Namprempre, Phillip Rogaway, and Tom Shrimpton. AE5 security notions. 2013.

[60] Andrey Bogdanov, Martin M Lauridsen, and Elmar Tischhauser. AES-Based Authenticated Encryption Modes in Parallel High-Performance Software. *IACR Cryptology ePrint Archive*, 2014:186, 2014.

[61] `http://ascon.iaik.tugraz.at/implementation.html`.

[62] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, and Sumio Morioka. CLOC: Authenticated encryption for short input. In *Fast Software Encryption*, pages 149–167. Springer, 2014.

[63] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi. SILC: SImple Lightweight CFB. *CAESAR submission*, 2014.

[64] `https://github.com/gvanas/KeccakCodePackage`.

[65] `http://pi-cipher.org/`.

[66] Hristina Mihajloska, Mohamed El Hadedy, and Kevin Skadron. Lightweight version of π-cipher. 2015.

[67] `http://primates.ae/implementation/`.

[68] Avik Chakraborti, Anupam Chattopadhyay, Muhammad Hassan, and Mridul Nandi. TriviA: A Fast and Secure Authenticated Encryption Scheme. In *Cryptographic Hardware and Embedded Systems–CHES 2015*, pages 330–353. Springer, 2015.

[69] `https://bench.cr.yp.to/supercop.html`.

[70] `http://www1.spms.ntu.edu.sg/~syllab/speed/`.

[71] Kris Gaj, Jens-Peter Kaps, Venkata Amirineni, Marcin Rogawski, Ekawat Homsirikamol, and Benjamin Y Brewster. ATHENa-automated tool for hardware evaluation: toward fair and comprehensive benchmarking of cryptographic hardware using FPGAs. In *2010 International Conference on Field Programmable Logic and Applications (FPL),*, pages 414–421. IEEE, 2010.

[72] Karim M Abdellatif, Roselyne Chotin-Avot, and Habib Mehrez. AEGIS-Based Efficient Solution for Secure Reconfiguration of FPGAs. In *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems*, pages 37–40. ACM, 2016.

[73] Makoto Kotegawa, Keisuke Iwai, Hidema Tanaka, and Takakazu Kurokawa. Optimization of Hardware Implementations with High-Level Synthesis of Authenticated Encryption. *Bulletin of Networking, Computing, Systems, and Software*, 5(1):26–33, 2016.

[74] Markku-Juhani Olavi Saarinen. Simple AEAD hardware interface (SÆHI) in a SoC: Implementing an on-chip Keyak/WhirlBob coprocessor. In *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices*, pages 51–56. ACM, 2014.

[75] `http://www1.spms.ntu.edu.sg/~diac2015/program.shtml`.

[76] Joseph Salowey, Abhijit Choudhury, and David McGrew. AES Galois Counter Mode (GCM) cipher suites for TLS. Technical report, 2008.

[77] David McGrew and D Bailey. AES-CCM cipher suites for transport layer security (TLS). Technical report, 2012.

[78] Helena Handschuh and Bart Preneel. Key-recovery attacks on universal hash function based MAC algorithms. In *Annual International Cryptology Conference*, pages 144–161. Springer, 2008.

[79] Daniel J Bernstein. Caesar: Competition for authenticated encryption: Security, applicability, and robustness, 2014.

[80] Farzaneh Abed, Christian Forler, and Stefan Lucks. General overview of the first-round caesar candidates for authenticated ecryption. *IACR ePrint*, 792:2014, 2014.

[81] `https://cryptography.gmu.edu/athenadb/`.

[82] P Rogaway and T Shrimpton. Deterministic Authenticated-Encryption. In *Advances in Cryptology–EUROCRYPT*, volume 6, 2007.

[83] Tetsu Iwata and Yannick Seurin. Reconsidering the security bound of aes-gcm-siv. *IACR Transactions on Symmetric Cryptology*, 2017(4):240–267, 2017.

[84] `https://cyber.biu.ac.il/aes-gcm-siv/`.

[85] Shay Gueron, Adam Langley, and Yehuda Lindell. AES-GCM-SIV: Specification and Analysis. *IACR Cryptology ePrint Archive*, 2017:168, 2017.

[86] Farzaneh Abed, Scott Fluhrer, Christian Forler, Eik List, Stefan Lucks, David McGrew, and Jakob Wenzel. Pipelineable on-line encryption. In *International Workshop on Fast Software Encryption*, pages 205–223. Springer, 2014.

[87] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Deoxys v1. 3, 2015. Second-round submission to the CAESAR competition.

[88] Jérémy Jean, Ivica Nikolic, Thomas Peyrin, and Yannick Seurin. Deoxys v1. 41. *Submitted to CAESAR*, 2016.

[89] Elena Andreeva, Begül Bilgin, Andrey Bogdanov, Atul Luykx, Florian Mendel, Bart Mennink, Nicky Mouha, Qingju Wang, Kan Yasuda, and D Compute. Submission to the caesar competition. *PRIMATEs*, 1:01, 2014.

[90] David Canright. A very compact S-box for AES. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 441–455. Springer, 2005.

[91] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A compact Rijndael hardware architecture with S-box optimization. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 239–254. Springer, 2001.

[92] Atri Rudra, Pradeep K Dubey, Charanjit S Jutla, Vijay Kumar, Josyula R Rao, and Pankaj Rohatgi. Efficient Rijndael encryption implementation with composite field arithmetic. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 171–184. Springer, 2001.

[93] Xinmiao Zhang and Keshab K Parhi. On the optimum constructions of composite field for the AES algorithm. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 53(10):1153–1157, 2006.

[94] Zheng Yuan, Yi Wang, Jing Li, Renfa Li, and Wei Zhao. FPGA based optimization for masked AES implementation. In *IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS), 2011*, pages 1–4. IEEE, 2011.

[95] Paolo Maistri, Sébastien Tiran, Philippe Maurine, Israel Koren, and Régis Leveugle. Countermeasures against EM analysis for a secured FPGA-based AES implementation. In *International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2013*, pages 1–6. IEEE, 2013.

[96] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, and Thomas Unterluggauer. ISAP–Towards Side-Channel Secure Authenticated Encryption. *IACR Transactions on Symmetric Cryptology*, 2017(1):80–105, 2017.

[97] Sai Zhang and Naresh R Shanbhag. Embedded Algorithmic Noise-Tolerance for Signal Processing and Machine Learning Systems via Data Path Decomposition. *IEEE Transactions on Signal Processing*, 64(13):3338–3350, 2016.

[98] Kyung-Ju Cho, Kwang-Chul Lee, Jin-Gyun Chung, and Keshab K Parhi. Design of low-error fixed-width modified booth multiplier. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(5):522–531, 2004.

[99] S Hamid Nawab, Alan V Oppenheim, Anantha P Chandrakasan, Joseph M Winograd, and Jeffrey T Ludwig. Approximate signal processing. *Journal of VLSI signal processing systems for signal, image and video technology*, 15(1-2):177–200, 1997.

[100] Jongsun Park, Jung Hwan Choi, and Kaushik Roy. Dynamic bit-width adaptation in DCT: an approach to trade off image quality and computation energy. *IEEE transactions on very large scale integration (VLSI) systems*, 18(5):787–793, 2010.

[101] Charbel Sakr, Ameya Patil, Sai Zhang, Yongjune Kim, and Naresh Shanbhag. Understanding the energy and precision requirements for online learning. *arXiv preprint arXiv:1607.00669*, 2016.

[102] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of human genetics*, 7(2):179–188, 1936.

[103] Jie Han and Michael Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *2013 18th IEEE European Test Symposium (ETS)*, pages 1–6. IEEE, 2013.

[104] Mohammed Shoaib, Niraj Jha, and Naveen Verma. A low-energy computation platform for data-driven biomedical monitoring algorithms. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 591–596. IEEE, 2011.

[105] Duckhwan Kim, Jaeha Kung, and Saibal Mukhopadhyay. A Power-Aware Digital Multilayer Perceptron Accelerator with On-Chip Training based on Approximate Computing. *IEEE Transactions on Emerging Topics in Computing*, 2017.

[106] Amr Suleiman and Vivienne Sze. Energy-efficient HOG-based object detection at 1080HD 60 fps with multi-scale support. In *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pages 1–6. IEEE, 2014.

[107] Mohsen Imani, Max Masich, Daniel Peroni, Pushen Wang, and Tajana Rosing. CANNA: Neural network acceleration using configurable approximation on GPGPU. In *Design Automation Conference (ASP-DAC), 2018 23rd Asia and South Pacific*, pages 682–689. IEEE, 2018.

[108] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[109] Robert E Schapire. Theoretical views of boosting and applications. In *Algorithmic Learning Theory: 10th International Conference, ALT'99, Tokyo, Japan, December 1999. Proceedings*, page 13. Springer, 1999.

[110] Sayed Ahmad Salehi, Rasoul Amirfattahi, and Keshab K Parhi. Pipelined architectures for real-valued FFT and hermitian-symmetric IFFT with real datapaths. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 60(8):507–511, 2013.

[111] Joseph M Winograd and S Hamid Nawab. Incremental refinement of DFT and STFT approximations. *IEEE Signal Processing Letters*, 2(2):25–27, 1995.

[112] Joseph M Winograd, S Hamid Nawab, and Alan V Oppenheim. FFT-based incremental refinement of suboptimal detection. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996*, volume 5, pages 2479–2482. IEEE, 1996.

[113] Seogoo Lee and Andreas Gerstlauer. Fine grain word length optimization for dynamic precision scaling in DSP systems. In *Very Large Scale Integration (VLSI-SoC), 2013 IFIP/IEEE 21st International Conference on*, pages 266–271. IEEE, 2013.

[114] Varun Bajaj and Ram Bilas Pachori. Epileptic seizure detection based on the instantaneous area of analytic intrinsic mode functions of EEG signals. *Biomedical Engineering Letters*, 3(1):17–21, 2013.

[115] Lalit M Patnaik and Ohil K Manyam. Epileptic EEG detection using neural networks and post-classification. *Computer methods and programs in biomedicine*, 91(2):100–109, 2008.

[116] Qi Yuan, Weidong Zhou, Yinxia Liu, and Jiwen Wang. Epileptic seizure detection with linear and nonlinear features. *Epilepsy & Behavior*, 24(4):415–421, 2012.

[117] Yun Park, Lan Luo, Keshab K Parhi, and Theoden Netoff. Seizure prediction with spectral power of EEG using cost-sensitive support vector machines. *Epilepsia*, 52(10):1761–1770, 2011.

[118] Zisheng Zhang and Keshab K Parhi. Seizure prediction using polynomial SVM classification. In *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*, pages 5748–5751. IEEE, 2015.

[119] Zisheng Zhang and Keshab K Parhi. Low-complexity seizure prediction from iEEG/sEEG using spectral power and ratios of spectral power. *IEEE transactions on biomedical circuits and systems*, 10(3):693–706, 2016.

[120] Zisheng Zhang and Keshab K Parhi. Seizure detection using regression tree based feature selection and polynomial SVM classification. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 6578–6581. IEEE, 2015.

[121] Adam Page, Siddharth Pramod Tim Oates, and Tinoosh Mohsenin. An ultra low power feature extraction and classification system for wearable seizure detection. In *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*, pages 7111–7114. IEEE, 2015.

[122] Manohar Ayinala and Keshab K Parhi. FFT architectures for real-valued signals based on radix-2^3 and radix-2^4 algorithms. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(9):2422–2430, 2013.

[123] Wei-Hsin Chang and Truong Q Nguyen. On the fixed-point accuracy analysis of FFT algorithms. *IEEE Transactions on Signal Processing*, 56(10):4673–4682, 2008.

[124] Tran Thong and Bede Liu. Fixed-point fast Fourier Transform Error Analysis. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(6):563–573, 1976.

[125] Randall B Perlow and Tracy C Denk. Finite wordlength design for VLSI FFT processors. In *Conference Record of the Thirty-Fifth Asilomar Conference on Signals, Systems and Computers, 2001.*, volume 2, pages 1227–1231. IEEE, 2001.

[126] Mojtaba Bandarabadi, César A Teixeira, Jalil Rasekhi, and António Dourado. Epileptic seizure prediction using relative spectral power features. *Clinical Neurophysiology*, 126(2):237–248, 2015.

[127] Upenn and mayo clinic's seizure detection challenge. USA. `https://www.kaggle.com/c/seizure-detection`.

[128] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd International conference on Machine Learning*, pages 625–632. ACM, 2005.

[129] Peter D Welch. The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *IEEE Transactions on Audio and Electroacoustics*, 15(2):70–73, 1967.

[130] Keshab K Parhi and Manohar Ayinala. Low-complexity Welch power spectral density computation. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(1):172–182, 2014.

[131] Adam Page, Chris Sagedy, Emily Smith, Nasrin Attaran, Tim Oates, and Tinoosh Mohsenin. A flexible multichannel EEG feature extractor and classifier for

seizure detection. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(2):109–113, 2015.

[132] Sang-Min Kim, Jin-Gyun Chung, and Keshab K Parhi. Low error fixed-width CSD multiplier with efficient sign extension. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 50(12):984–993, 2003.