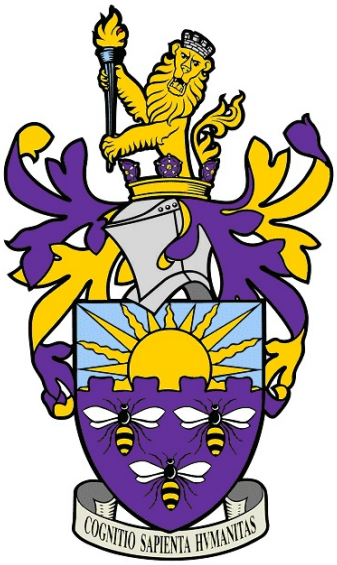
The University of Manchester
School of Computer Science



Crypto library for vectorised FPGA interlays

Anuj Vaishnav
Supervised by Dirk Koch

A report submitted in part fulfilment for the degree of
BEng (Hons) in Computer System Engineering
May 2017

Abstract

As a solution to the rising of Dark Silicon areas and to the need of adding more accelerators and application specific instructions in future CPUs, this project examines the idea of adding a run-time reconfigurable FPGA-like fabric (called an FPGA Interlay) directly into the CPU core to allow application-specific customization in the field. This project presents the implementation and trade-offs involved for cryptography cores as instruction set extensions for ARM A9 CPU where the NEON vector unit is replaced with a fine-grained Interlay fabric of identical size and vector interface. The results of this study show that this Interlay is 7.7x faster for AES than the hardened NEON vector unit and can provide functionality that is not available on NEON. Additionally, this project highlights possible techniques for optimising the FPGA Interlay and the NEON vector unit.

Acknowledgements

Firstly, I would like to thank my supervisor, Dirk Koch, for his continuous support and guidance throughout the course of the project – without which my scope of achievement with this project would have been very limited. Secondly, I would like to thank my family and friends for their help and encouragement through the difficult times. Lastly, I would like to express gratitude towards my second marker, Richard Neville, for his insights and comments during the seminar and demonstration stages of this project.

All your help and support is deeply appreciated!

‘Be curious and keep moving forward.’
Walt Disney

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter provides a short overview of the motivation and problem this project contributes towards, followed by the aims and objectives. Finally, it presents an overall structure of the report.

1.1 Motivation

Modern day processors currently rely on Moore's law [1] of transistor scaling to provide more functionality by replicating cores [2] or by using individual application specific acceleration units, e.g., MMX [3] and NEON [4]. However, this will become involved as we approach smaller and smaller transistors. Further, due to the end of Dennard scaling [5] the power consumption will limit the growth in this direction as the amount of circuitry capable of being switched on simultaneously at a given thermal design power constraint (TDP) would decrease. This particular phenomenon of Dark Silicon has been studied and predicted to be the most important design factor for future CPU designs by Hadi Esmaeilzadeh et al. in [6].

Moreover, in the future, the new of Internet of things (IoT) paradigm would tend to be highly heterogeneous and low power [7]. Further, the requirements of secure communication among these devices imply a heavy use of cryptography and compression. The encryption algorithms and standards are periodically updated to retain a better security resistance. It means adding more individual units to accelerate application on these devices is going to be infeasible as chip design cost would get higher due to their niche requirements with comparatively smaller life cycle because of update requirements.

Consequently, a novel approach is required to use existing CMOS technology to increase performance and to support future applications. One such technique is to introduce a fine-grained FPGA[1] fabric inside a CPU core which is also known as an FPGA Interlay. It can be used to accelerate performance for various applications by providing run-time Instruction Set Extension (ISE) as needed and at a fixed cost of die area and power consumption by exploiting reconfigurable hardware. Further, it also provides other benefits like support for new architectural design and in-field updates without going again through the whole chain of chip design and

[1]Field programmable gate array (FPGA)

manufacturing, probably saving an order of magnitude of time and cost.

However, the development of an Interlay architecture is not studied in the past. Further, a conventional FPGA tends to require more amount of area and runs at lower clock frequency for a given design compared to an ASIC implementation[2] [8]. This is because of the logic requirements for supporting reconfigurable behaviour which adds additional functional units accompanied by multiple clock domains and a switching matrix. However, as an Interlay is supposed to be on-chip, it does not necessarily have to abide standard FPGA architecture. Thus, the requirements of an Interlay approach for delivering an ideal performance remains an open question. Consequently, this project, assumes certain characteristics of an Interlay and will be discussed in Section 2.1.

1.2 Aims and objectives

The aim of this project is to research and evaluate the potential performance achievable for cryptography using an Interlay approach and to identify and suggest possible features for an optimised Interlay design. To accomplish this aim, a detailed research has undertaken followed by the evaluation of the results and possible extensions. The procedure included investigation of various design factors and their trade-offs for a compact implementation with high throughput, followed by extensive verification and validation.

The objective of the project work has been to research and evaluate the following IP-cores for an Interlay approach:

- Symmetric Key Ciphers
  - Advance Encryption Standard (AES)
  - Data Encryption Standard (DES)
- Asymmetric Key Cipher Support
  - Montgomery multiplication
- Cryptographic Hash Functions
  - Secure Hashing Algorithm 1 (SHA1)
  - Secure Hashing Algorithm 2 (SHA2)
- Key Generation
  - True Random Number Generator (TRNG)

[2]Application Specific Integrated Circuit (ASIC) is a microchip designed for specific application.

1.3 Report Outline

The remainder of the report is structured as follows:

- **Chapter 2** provides the context on Cryptography primitives and Interlays.
- **Chapter 3** discusses the implementation and their trade-offs for all implemented IP-cores.
- **Chapter 4** focuses on the verification and evaluation of the IP-cores followed by suggestions for an optimised Interlay fabric and vector processor.
- **Chapter 5** summarises the project and highlights future work and experience of the project in hindsight.

Chapter 2

Background

This chapter will introduce FPGA Interlays in detail which acts as the requirement specification of this project, and it will further provide background information on cryptography primitives and algorithms that were chosen in this study for implementation.

2.1 FPGA Interlay

Modern processors feature specialised vector units which requires a high amount of silicon area among other ISEs. For example, the NEON vector unit for ARM Cortex-A9 consumes approximately 20% of the System-on-Chip (SoC) area which maps to about 1040 Look-up tables (LUTs), 8 Digital Signal Processing (DSP) and 4 Block RAM (BRAM) in FPGA primitives [9]. This study takes these resources as a guideline for the introduction of FPGA on-chip for modern processors. This available area is smaller compared to a standard FPGA, which makes it difficult for designing the IP-cores for it.

A conventional FPGA implements a range of primitives and clock domains at the cost of power and high frequency to support a wide range of digital designs. However, an Interlay fabric does not necessarily have to be a conventional FPGA as it only required to run on a single clock domain, preferably CPU frequency. Further, it will only have routing and primitives selected for the most common applications in both hardened and soft logic to lower the latency, power and area. The selection of the FPGA primitives and routing required for cryptography applications is performed by finding the common features among various IP-cores designed for the above resource requirements and identifying the potential performance achievable for them as discussed further in Chapter 4.

Moreover, there is a certain amount of hardened logic which is assumed to exist for the Interlay including the interface connection with the CPU, the Register file and the load-store unit. The interface considered for this study is fundamentally similar to the interface of a NEON vector unit of an ARM-A9 core. The interface provides the two 128-bit operands as inputs, one 128-bit result and 32-bit of instruction encoding. More details on instruction encoding and accessibility of operands are discussed in Section 3.2.1. Further, some applications would also require the ability

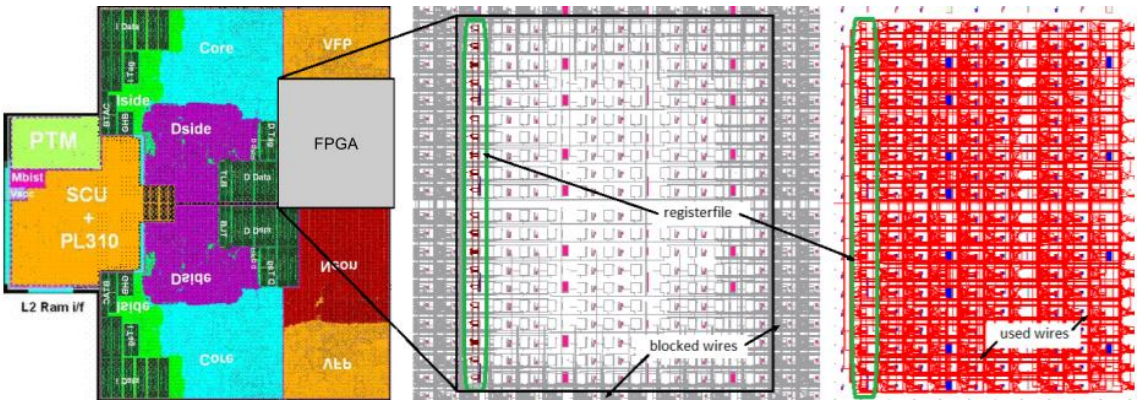


Figure 2.1: ARM Cortex A-9 Macrocell Floorplan edited to highlight FPGA Interlay concept on a SoC [10].

to stream instructions to identify which operation needs to be performed. The stream is provided by the hardened Fetch unit. For this study, we assume the hardened units to be identical with NEON regarding latency and behaviour. Figure 2.1 shows how an Interlay can be placed on a SoC and how it would be occupied by the IP-cores when in use.

2.2 Cryptographic Primitives

There are 4 different type primitives required for providing a cryptographic solution in the current real-life applications like OpenSSL [11], NSA communication [12] and banking connections [13]. These are symmetric ciphers, asymmetric ciphers, cryptographic hashing and key generation. The following subsection discusses them in detail and reasons the selection of algorithms chosen for this study for each component.

2.2.1 Symmetric Ciphers

Symmetric ciphers operate with common knowledge of the secret key shared between the parties involved in communication, for encrypting a large message efficiently. There are two types of Symmetric ciphers: Stream and Block ciphers. Block ciphers work on a fixed size of input at a time, while stream ciphers operate on a continuous stream of data. However, with an appropriate mode of operations, all block-ciphers can be used to construct a stream cipher, which is commonly applied. Hence, this project focuses only on block-ciphers only.

Conventionally, a block-cipher is created from a network of substitution and per-

mutation operations to improve security at relatively low-cost [14]. A permutation is an act of *reordering* the input bits with a bijective function to create unrecognisable output also known as ciphertext. While substitution is an act of *replacing* the input text bits to ciphertext bits. Further, permutations are mapped to hardware by shift and rotate operations, whereas, substitution is commonly performed by the utilisation of look up tables with associated memory (also known as S-box). Furthermore, a permutation operation on a CPU is expensive to perform in terms of time and area requirements as it would require barrel shifters which are not always provided in micro-controllers. Similarly, S-box is expensive because memory needs a magnitude of power and area. Hence, both operations represent a strong challenge for a cost sensitive systems.

Current NIST standard for symmetric block-ciphers are Advance Encryption Standard (AES) [15] and Triple Data Encryption Standard also known as Triple DES or TDES [16]. The Triple DES algorithm is the application of Data Encryption Standard (DES) [17] with repeated encryption and decryption using different keys. DES is also used in many different variants to encrypt and hash data due to its strong mixing property. This study chose to implement both AES and DES in the IP-core library to maximise the utility and test the Interlay approach for real-life applications.

2.2.2 Asymmetric Ciphers

Asymmetric ciphers or Public-key cryptography uses a pair of keys: a *public key* which can be made public and a *private key* which is only known to the owner. They often achieve security by relying on mathematically hard problems to solve such as discrete logarithm, integer factorization and elliptic-curve relationship with cyclic group properties. These cryptosystems can be used to do authentication or encryption, but mostly they are used for authentication and secure transportation and distribution of symmetric keys as they are computationally expensive. The most commonly used algorithms in the order of mathematical problem described are Digital Signature Algorithm (DSA) [18], RSA [19] and Elliptic curve cryptography [20] respectively.

The building block and bottleneck in the performance of all the above mathematically hard problems are modular multiplication on extended long numbers. The most recommended approach for this operation is Montgomery modular multiplication [21] due to its efficiency on computers. Thus, this study focuses on the implementation of Montgomery multiplication to support all type of Asymmetric ciphers, and the algorithm chosen for this application is discussed in Section 3.1.2.

2.2.3 Cryptographic Hashing

The cryptographic hash functions are a subclass of hashing functions with certain unique properties including:

- **One-way property** (preimage resistant):
  H(x) is easy to compute for any given x, but it is hard to calculate x for a given h, such that H(x) = h.
- **Weak collision resistance** (2^{nd} preimage resistant):
  Given x, it is hard to find y $\neq$ x, such that H(y) = h.
- **Strong collision resistance** (collision resistance):
  It is hard to find two different messages, x and y such that H(y) = H(x).

These functions are an integral part of various applications such as verifying the integrity of a file or message, Proof-of-work systems, Password verification and Pseudo-random generation and key derivation. The current NIST standard recommends Secure Hash Algorithm (SHA) family [22] for hashing which constitutes of SHA-1, SHA-2 and SHA-3 with varying lengths. These algorithms are made up of multiple rounds over which the input is consumed gradually with the help of expansion and compression phases. Figure 2.2 highlights the kernel and the steps in the SHA-1 algorithm. Wherein the A, B, C, D, E are part of the state, F is a non-linear function that varies, W_t is message block of round t and K_t is a round constant of round t.

These algorithms represent an interface challenge because the required input size for the hash computation exceeds the size allowed by the interface of an Interlay in a single clock cycle. Resulting in lower throughput unless an appropriate solution is applied. In this study, only SHA-1 and SHA-2 are chosen as a wide variety of applications already rely on them and time constraints for the project do not allow a thorough research on SHA-3 implementation as well.

2.2.4 Key generation

A secure key generation is important because every encryption is only as strong as the key used for it. Most of the successful attack on NIST standard algorithms are due to the weakness of key generation, allowing the attacker to reduce the search space for the key. Multiple key generation methods exist out of which NIST recommends the use of cryptographically secure Pseudo-random Number Generator (PRNG) and True Random Number Generator (TRNG) [23].

A PRNG is a deterministic way to generate a number which appears to be random under statistical analysis and uses a seed to exploit the system entropy for better results. However, if the seed can be identified or manipulated in any

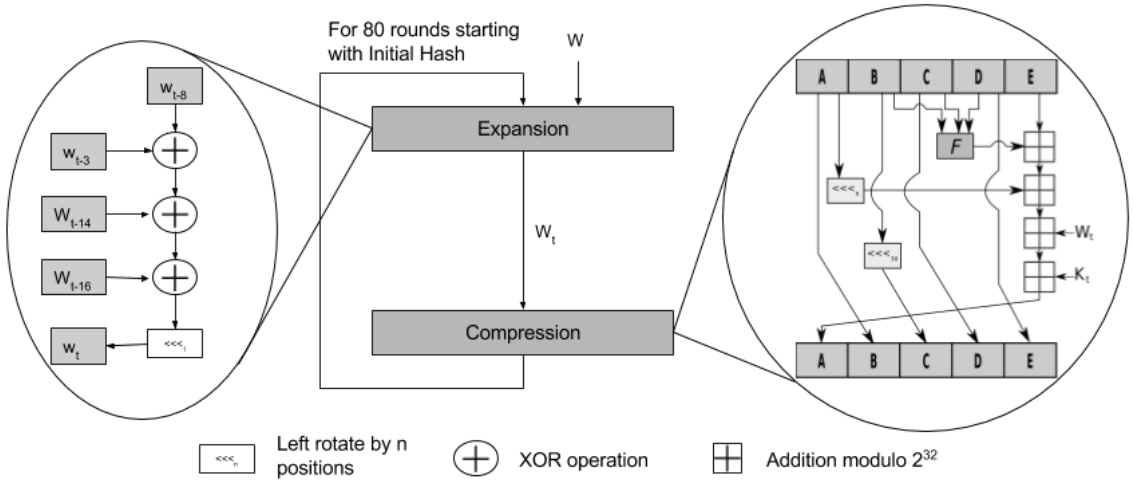


Figure 2.2: SHA1 algorithm structure.

manner, the key generation pattern can be cracked, leaving the security of the whole encrypted system compromised. TRNG addresses this issue by using a physical phenomenon with statistical randomness to generate a random number. Often they are based on microscopic phenomena that generate statistically random noise signals, such as electromagnetic phenomena and thermal noise. The statistical randomness of a TRNG can be further improved by using its outcome to seed a PRNG or by applying a strong mixing function to remove any possible correlation in the source. Hence, in this project, a TRNG is constructed using a random physical source and a strong mixing function as further discussed in Section 3.1.4.

2.2.5 Block-cipher mode of operation

A mode of operation is a method of using the block-ciphers to achieve confidentiality or authentication. It describes how to securely generate an amount of data larger than a block size, by recurrently applying cipher's single-block operation [24]. Multiple modes of operation exist however the most common methods for encryption are Electronic Codebook (ECB), Cipher Block Chaining (CBC) and Counter mode (CTR).

The simplest mode is ECB. Wherein the message is divided into multiple blocks and encrypted independently of each other. The result is then concatenated to form the ciphertext of the message, as shown in Figure 2.3.

Similarly, CBC mode also splits the input into blocks, but it orders them and creates a feedback by XORing the input text with the outcome of the previous block, as shown in Figure 2.4. The first block uses an initialization vector of user's

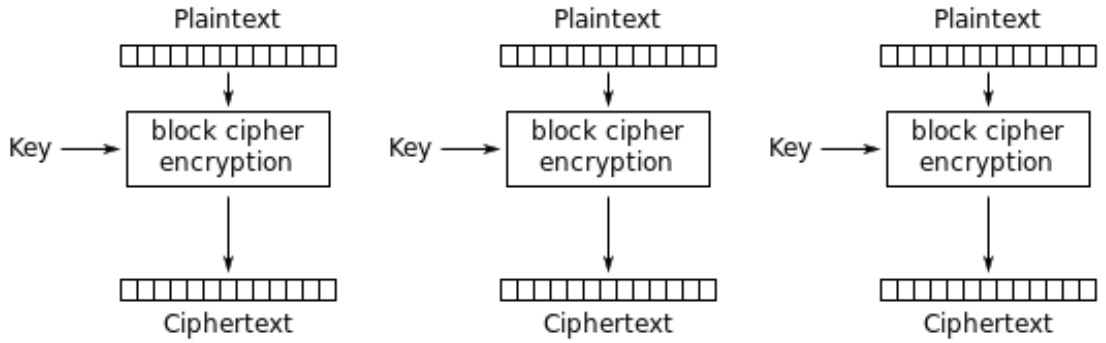


Figure 2.3: Electric Codebook (ECB) mode of encryption.

choice, to ensure the consistency of security. This mode provides better protection due to its feedback loop. However, it leads to inferior performance due to serialised processing.

The CTR mode operates differently from the other modes and is often used to construct a stream cipher based on a block-cipher. It encrypts a counter value and XORs the result with input text to generate ciphertext, creating a continuous stream of encrypted data as each block can be pre-generated and concurrently due to its independent nature. CTR is highly secure as each block operates on a different counter value which leads to the ever changing output. Therefore, is widely accepted [25].

2.3 Related Work

This section discusses the related work on FPGA architectures, custom instruction selection and the hardware implementation of the algorithms chosen in Section 2.2.

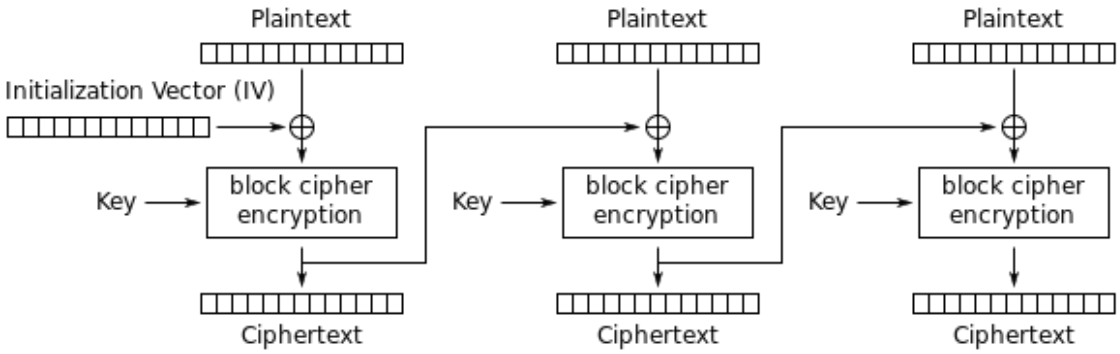


Figure 2.4: Cipher Block Chaining (CBC) mode of encryption.

2.3.1 FPGA architecture

The concept of accelerating applications with reconfigurable functional units have been studied extensively in the past. The classic example of this is Chimaera [26]. However, this technique is limited to a small amount of area and would not be able to support complex operations as a result.

Moreover, the unconventional approach is combining Application Specific Instruction Processor architecture with reconfigurable parts for specific applications such as cryptography [27]. These processors though do not extend easily to other domains as their reconfigurable behaviour is limited and contains certain hardened logic particular to the application domain.

Conventionally for customising CPU instruction sets on FPGAs is done by implementing a complete processor on FPGA, known as soft-cores. These soft-cores offers high flexibility and extendability compared to processors made with ASIC technology. Hence, they have been studied extensively for various specific applications [28, 29, 30, 31, 32]. Specifically, architectures based on NIOS II range from Cryptography [33] to Image processing [28] to Audio applications [34]. Vector units designed based on soft-cores with FPGA-specific optimisation have been shown to benefit drastically [35]. Common approaches focus on a comparatively slow soft CPU with an interface to extend a scalar CPU [26, 36]. Contrarily, his project examines only the reconfigurable parts that promise great potential to improve performance considering a hardened CPU leading to a more compact and high-performance design.

2.3.2 Selection of Custom Instructions

The ISE for FPGAs is often generated by the mapping the intensively used code-fragments to the custom instructions [37, 38]. Other techniques used include operator-chaining and ISA subsetting. Operator chaining avoids the register write-back stage while ISA subsetting supports only the instructions required by the kernel at a given time for saving FPGA resources. Examples of these include [39, 40, 29, 41].

Furthermore, selected instructions can also be generated by directly converting a time-consuming software function into a Hardware descriptions by utilising high level synthesis (HLS) tools [42] to reduce the development time, but the efficiency of the generated design may not be optimal for all types of use cases with this approach.

All of the above approaches can be applied collectively to keep the resource requirements low for an Interlay. However, none of them would be likely able to extract the speed up as good as a handcrafted design based on the domain knowledge

and different abstraction level of logic.

2.3.3 FPGA implementations of Cryptography

Due to Cryptography's importance in computer science, many FPGA implementations exist in the literature. The following sections categorise them in 4 different cryptography primitives and highlight the work done for the chosen algorithms.

Symmetric ciphers

The implementations of AES and DES in literature tend to focus on either high throughput [43, 44] or small area [45, 46], which are contrasting in nature. Small area designs typically take a specialised processor approach and operate at byte level to keep resources under few hundred LUTs. This method commonly calculates S-box values instead of using memory [45, 47]. On the contrary, high throughput designs implement substitution operation (S-box) as look up tables, employ deep pipelining and round unrolling. One of the most efficient configuration for AES requires the use of 10 unrolled rounds, BRAM for S-box and a two-stage round pipeline [43]. However, this configuration requires 3766 LUTs which is not possible in this study, due to the area constraint of 1040 LUTs.

SHA

The common techniques employed for fast hashing algorithm implementation are operation rescheduling, re-timing, pipelining and loop unrolling [48, 49, 50]. However, these methods increase resource consumption drastically and hence, many applications avoid them and focus on resource requirements. Compact designs tend to use smaller datapath widths and a custom ALU approach and achieve a design as small as 139 LUTs but at the expense of low throughput [51].

Other than minimization of resources and maximisation of performance, literature has a little amount of work on optimising symmetric ciphers and cryptographic hash functions for particular resource bounds. However, this is needed for Interlays so they can provide the best performance from a fixed and small set of resources.

Modular Multiplication

Modular multiplication implementation on FPGAs is usually performed using either Interleaved Modular Multiplication (IMM) or Montgomery Modular Multiplication (MMM). IMM relies on repeated additions and reduction of partial products. Further, there are many versions of IMM which target a serial or parallel implementation

by employing Booth Encoding, higher radix or Carry Save Adders to achieve better performance and area [52, 53, 54]. On the other hand, MMM is based on converting the division to simple shift operations and has gone through many modifications since its first proposal. Many different Operand Scanning strategies for MMM are discussed in [55] and other higher radix based implementations are shown in [56, 57]. Further, [58] has that MMM is more efficient with a given amount of hardware compared to IMM in and therefore is a more applicable for an Interlay. However, the use of higher radix and Booth Encoding tends to require a large area and are not suitable for the Interlay constraints.

Random number generator

The random number is obtained on FPGAs by utilisation of PRNG and TRNG units [59]. The conventional source of randomness are ring oscillators and will be further explained in Section 3.1.4. However, there is no common trend on specific functions used to amplify the randomness after the seed production. The choice of mixing function and PRNG tends to vary from Blum Blum Shub [59] to specific hash functions and block-ciphers [60] to spatiotemporal chaos [61]. Therefore, this study will use its implementation of a strong mixing function based on the resource constraints.

Chapter 3

Design and Implementation

This chapter provides a short description of all IP-core implementations and trade-offs made for them. Further discussion on their exact performance, resource requirements and evaluation are done in Chapter 4.

3.1 Cryptographic IP-cores

All implemented Interlay instructions produce a single round result to avoid storage of any state *inside* the reconfigurable fabric otherwise this would increase the context switch time. Further, it provides flexibility of deviating and extending from the standard implementation of algorithms (e.g., using a different number of rounds).

It also maximises the utilisation of vector interface for certain cores (e.g., AES). The key expansion is not implemented in the cores to reduce the resource requirements, as it is possible to load the sub-keys into the register file earlier and reuse them multiple times. Lack of key expansion logic also allows IP-cores to be used for different key widths of symmetric cyphers, enabling more customisation at the user level. The following subsections describe how the best throughput under resource constraints has been achieved for each type of IP-core. However, there are other variants of the cores which are available in the library to use with different timing behaviour, and power requirements and their respective performances as described in Chapter 4.

3.1.1 Symmetric Ciphers

The library supports all commonly used types of block-cipher mode of operations for block encryption, stream cipher and authentication as it operates at the round level. The specific implementation details for each cipher core are discussed below.

AES

A datapath of 128-bit is selected as it maximises utilisation of the vector interface of an Interlay and as it is a natural fit for the working set of AES [15]. It also allows the mapping of the shift-rows stage of AES algorithm to simple wiring, avoiding the costly shift registers and long latencies. AES reuses the same kernel in all rounds

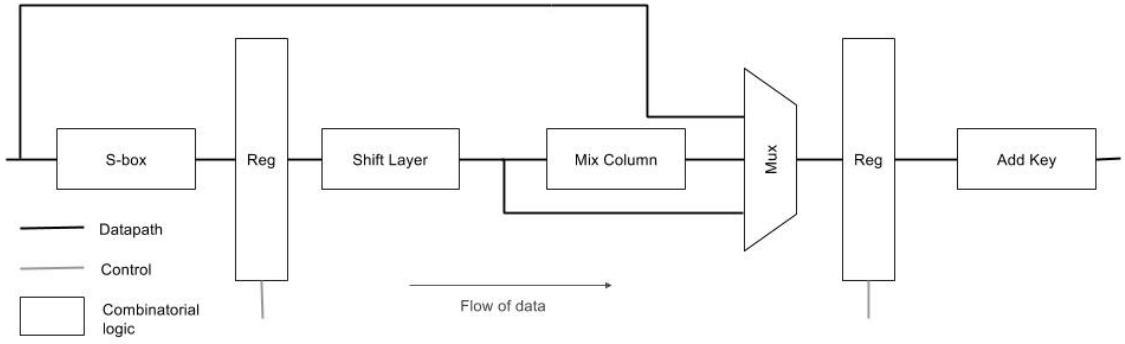


Figure 3.1: The AES Encryption datapath for 3 stage pipeline.

with exceptions of initial and last rounds. Initial round requires an additional key whitening step at the start, while final round skips the Mix-Column stage of a normal round. These two different behaviours share the operation with normal round operations. Hence these are offered as three separate instructions for better resource utilisation at the cost of an additional multiplexer and normal round hardware necessities.

The throughput is improved by employing pipelining to split the standard round into three stages at the expense of two additional 128-registers. The S-box is implemented with BRAM instead logic blocks to achieve a compact size and lower logic block resources. Further, it is shown to be the most efficient technique for improving AES performance at round level [43]. Note, S-box implementation does not share the look-up tables, all the 128-bits are substituted concurrently for better throughput. Additionally, the design is dynamic in nature for S-box instantiation, as it allows switching between BRAM and logic based on the availability of BRAM blocks. It allows the design to adapt to any resource constraint of reconfigurable fabric.

Moreover, the use of round unrolling is avoided on purpose despite its proven ability to improve performance in [43], as it exceeds the resource constraints. Thus the above techniques are affordable ways to improve throughput without violating the constraints.

DES

The core is made up of combinatorial logic as the latency of most logic of DES is reduced by other techniques, hence making pipelining latency considerably expensive with low benefits. A 64-bit datapath is used to keep the resources to a minimum as DES only uses 64 bits at a time [62]. The main round is implemented

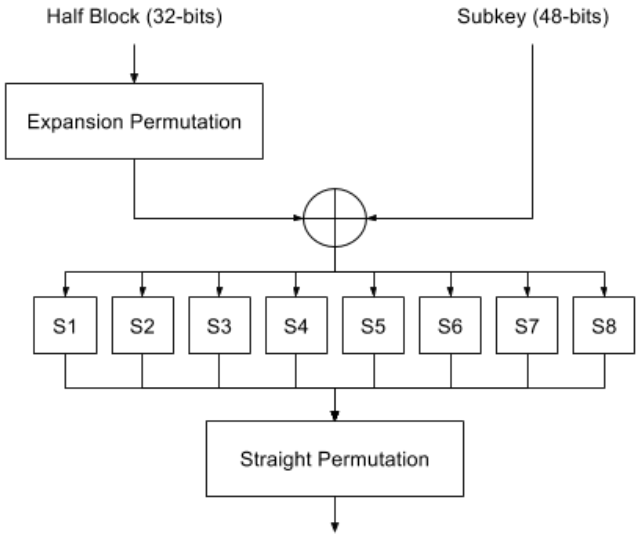


Figure 3.2: The Feistel function of DES.

in the core i.e. without initial and final permutation to avoid storing the state and for supporting various non-standard versions of the algorithm as its often used in password authentication and hashing with modification, for example, [63, 64]. It also means that only a single instruction is required to implement the majority of the DES operations. To achieve the most efficient resource saving all the shifts and permutations of the Feistel function (shown in 3.2 as E and P blocks) are mapped to simple wiring, only leaving S-box (Look-Up tables) and XOR-gates as other operations. S-boxes are implemented in LUTs instead of BRAM because LUTs do not need a clock to evaluate a function. This results in the lowest possible latency which is better for DES performance. Moreover, DES needs only half the number of S-boxes than AES. The resulting module is smallest in the library and permits it to be coupled with any other unit without undergoing reconfiguration or to be used as a building block for other instructions.

Countermeasures Against Attacks

The above implementations provide resistance to the Cache, Timing and Simple Power Analysis (SPA) side-channel attacks. Cache attack is based on the ability to monitor cache accesses of the victim on a shared system and is avoided by usage of inbuilt S-boxes leaving only data and key schedule for import upon execution. The key schedule and data (once computed), can be kept in the register file throughout the encryption process making this type of attack difficult to succeed.

A timing attack relies on the ability to link the differences in time taken to perform operations with some knowledge about the secret. Usually, this results from

the conditional branches, varying micro-codes and optimisations performed by the compiler or programmer based on the data. The above designs do not contain any branches or bypass routes in any hardware components based on data and maintain the constant timing for all rounds and operations. Similarly, SPA is difficult as it is based on power variations of the operations: the constant timing and activity makes it difficult to identify any implicit knowledge about the secret.

3.1.2 Asymmetric Key Cipher Support

As discussed in Chapter 2, most Public-key systems work on cyclic groups and require modular multiplication. The implementation below describes a possible Montgomery Multiplication implementation for Interlay fabric for 128 bit.

Montgomery Multiplication

There are various versions of Montgomery multiplication in literature, but most of them use a multiplier as building block internally, which can be expensive to im-

Algorithm 1 Montgomery Multiplication [65]

Require: $X = \sum_{i=0}^{n-1} x_i \cdot 2^i$, $Y = \sum_{i=0}^{n-1} y_i \cdot 2^i$, $M = \sum_{i=0}^{n-1} m_i \cdot 2^i$, $0 \leq X, Y < M$
Ensure: $P = X \times Y \times 2^{-n}\ mod\ M$
  // n: number of bits in X
  // R: precomputed value of $Y + M$
  $S = 0; C = 0$
  for $i = 0$ to $n - 1$ **do**
    if $s_0 = c_0$ and $not(x_0)$ **then**
      $I = 0$
    else if $s_0 \neq c_0$ and $not(x_0)$ **then**
      $I = M$
    else if $not(s_0$ **xor** c_0 **xor** $y_0)$ and x_0 **then**
      $I = Y$
    else if $(s_0$ **xor** c_0 **xor** $y_0)$ and x_0 **then**
      $I = R$
    end if
    $S, C = S + C + I$
    $S = S\ div\ 2;\ C = C\ div\ 2$
  end for
  $P = S + C$
  if $P \geq M$ **then**
    $P = P - M$
  end if
  return P

plement concerning resources. The implementation in [58] is closest to resource constraints for an Interlay requiring about 1534 LUTs for 256 bits on Virtex-6 FPGA. Their design utilises the algorithm described in Algorithm 1 which avoids the use of multiplier directly and relies on a Carry Save Adder (CSA) for faster and less power consuming calculation. The implementation for our Interlay is based on the same algorithm using a Finite State Machine (FSM) and and a 128-bit datapath and it operates on 1 bit at a time, requiring 131 cycles for its computation. Going to higher radix would allow better performance but would consume more resources and requires a multiplier. The FSM is relatively straightforward and can be seen in Figure 3.3. The "Mult_loop" state makes use of a CSA to evaluate the result by differing the propagation of carry. This reduces the chip activity but does not shorten the critical path of the design, as carry must propagate at the end. The 128-bit carry propagation takes place in state "P_sum" and is the bottleneck of the design despite the use of optimised carry chain primitive called Carry4 from Xilinx. However, the performance is improved by early termination when the remaining bits to multiply are zero. Further, the design is parametrised allowing to choose the bit size of the unit, giving the option to use a smaller and faster core when a complete 128-bit multiplication is not required by the application, by shortening the carry chain which is the critical path.

3.1.3 Cryptographic Hash Functions

The implementations of hashing IP-cores do not include support for the final addition operation used in the algorithm to chain the outcome of more than one block. This is done to reduce the resource overhead of adders as they are only utilised at the

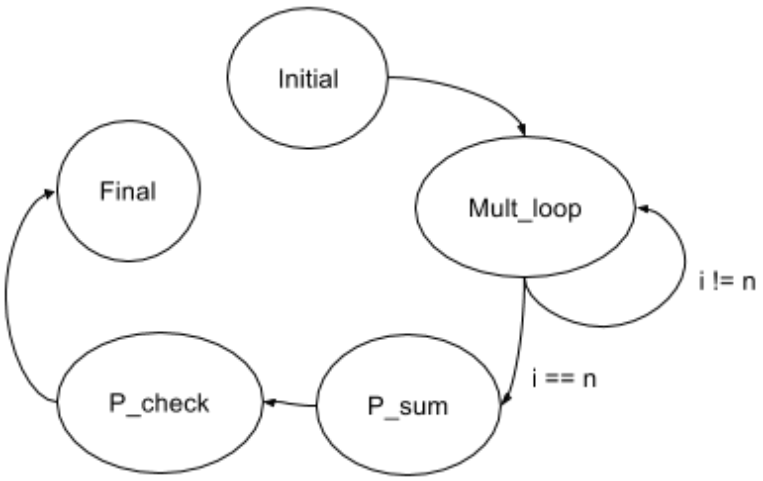


Figure 3.3: Finite State Machine of Montgomery Multiplier.

end of a block. It also allows custom implementations when throughput has higher priority than security, e.g., concatenating separate hashes to produce a non-standard fast-SHA implementation. The specific implementation details are discussed below.

SHA1

The SHA1 kernel consists of 2 main stages: message expansion and compression [66]. Expansion stage takes 16 32-bits-words and produces 80 32-bit-words. Given the interface constraints of our vector processor and no internal state, two 128-bit input operands cannot accommodate the message data for multiple rounds together with the previous hash. Hence, only the chunk of message relevant to the round is used at a time i.e. a single 32-bit-word. The core supports two instructions: Message expansion and compression, each with their own pipeline. Message expansion can be done just in time and just enough to keep the pipeline full for compression stage. The round information is stored in the instruction as a literal to allow selection of F and k values as shown in Figure 2.2. This suffice the input requirements while output requirements are satisfied by writing results over consecutive cycles. The shifts and rotations are mapped to wiring, avoiding the use of expensive barrel shifters. Pipelining and operation reordering is used to improve the critical path and throughput at low area cost.

SHA2

Similar technique as SHA-1 is applied to message expansion stage yet the input size required is higher than the interface allows. Hence, two coupled instructions are used to execute a single round whereby data is brought in and out in consecutive cycles. The input hash is first accessed and then the 32-bit message chunk, this allows to start processing the first cycle itself removing the waiting. Loop unrolling does not help to improve the throughput [50] thus 3-stage pipelining for a single round and operation reordering is used to improve throughput while keeping lower resource requirements. Implementation of k values by using wires tied to specific values allowing the tool to optimise area and resources well leading to space savings.

3.1.4 True Random Number Generator

The random number is derived as follows:
Let V be the reading from ring oscillators, $DES(x, k)$ be the result of single DES

round on data x with key k and $R[i]$ be the i-th random number generation.

$$R[0] = V$$
$$R[i] = (V \oplus DES(R[i-1], k))$$

The implementation is similar to the current NIST standard for PRNG using Triple DES [67]. The differences lie in the 1) seed used, 2) free running DES rounds and 3) generation of a 128-bit number by concatenation of two DES block results rather than a 64-bit number. Readings from 128 ring oscillators provide the natural source of randomness for each bit based on the electrical noise and clock jitter. This randomness then gets amplified by the strong mixing property of the DES round. The process repeats itself on every clock cycle with a feedback of the previous result via XOR gate after the initialization of the core, operating in a cipher block chaining mode. It results in the execution of 16 normal rounds of DES encryption, hence providing a very secure cryptographic random number with every cycle. The feedback loop is shown in Figure 3.4.

Ring oscillators of 3 NOT gates in length are chosen over other techniques as they are proven to work well as a source of randomness on FPGA devices with low resource requirement [68, 69]. The DES core discussed in Section 3.1.1 implements the DES round due to its low area and latency. Other block-ciphers can be employed potentially, including AES, but it would violate the resource requirements or lead to lower throughput if we implement a different version other than discussed above

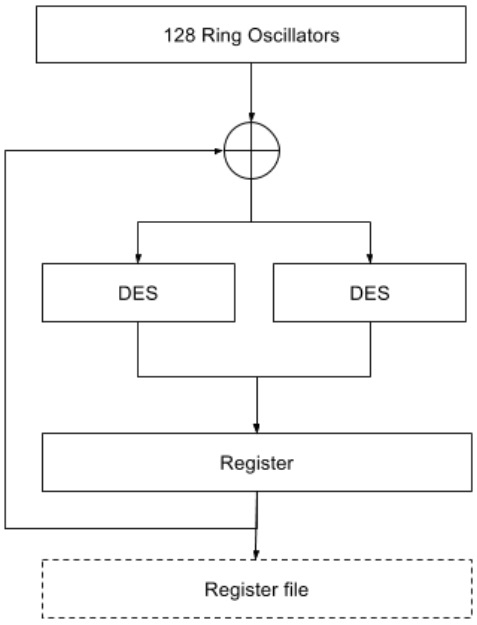


Figure 3.4: True Random Number Generator Schematic.

in the Interlay setting. Note, since a register is sampling a constantly changing signal of ring-oscillator, it may land in the meta-stable state. A meta-stable state arises from data hold time or setup time violations of required of a register. It is an unavoidable scenario but with a very low probability [70]. This probability can be made even smaller by adding an another register, as the probability reduces exponentially w.r.t. time [71]. The second register in this scenario is the register file.

3.2 Software

This section describes the software consideration and choices made to use the IP-cores mentioned above by first discussing the peculiarities of the instructions, followed by the interface exposed to the programmer and standard optimisation techniques employed in the library.

3.2.1 Instructions for IP-cores

Assuming the 32-bit instruction word is made available to the Interlay, the decode logic can be specific to the chosen configuration in the Interlay. For each configuration, a set of instruction encoding could be used to extract the maximum performance. This leads to multiple instructions and behaviours making the programming difficult. However, coupled with a software library the peculiarities of hardware are abstracted away. The assembly instructions in software library follow the same format as a normal vector instruction, containing the op-code and operand register codes with other additional bits which the configuration can use as required (shown in Figure 3.5). The register codes are accessed by the hardened logic while the IP-cores use the rest. The IP-cores mentioned above do not use all the bits available and it only needs a simple decode logic.

3.2.2 Software Library

The Interlay library is based on the combination of a hardware and software approach to maximise the potential performance and efficiency of the system as well

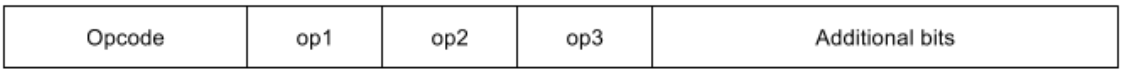


Figure 3.5: Custom instruction encoding format.

as usability. The software part has been written with the help of inline assembly. It employs all the standard optimisation techniques for each algorithm like function inlining, loop unrolling, precomputed tables and usage of arrays to exploit spatial data locality. Moreover, it gains more benefit from the software pipelining by scheduling data to avoid dependency hazards between the hardware pipeline stages. It relies heavily on the design knowledge of the hardware units used. The hardware pipeline depth limits the depth of software pipeline and thus is completely problem dependent. Hence, its application is not equally viable for all type of units. Further, software pipelining also implies independent data blocks and availability of many registers such that each stage can operate on separate data values concurrently. The register-file of Neon is only big enough to support this operation up to 3 stages for the IP-cores mentioned above, but the new 64-bit ARM architecture supports the double registers for extending the depth of pipeline in future.

However, the interface of the library does not expose the hardware behaviour to the programmer and is kept similar to OpenSSL [11] for better usability. It internally handles the lack of independent data for software pipelining by using NOP operations to avoid hazards. It also provides the various mode of operations for the block-ciphers and hashing algorithms, allowing the user to have better control over the trade-offs of security level and throughput of the system by deviating from the standard.

Chapter 4

Testing and Evaluation

This chapter focuses on the testing and evaluation of the implementation described in Chapter 3 by discussing the approaches used and by contrasting the performance and resources required for them. Finally, it provides suggestions on Interlay fabrics and the NEON instruction set.

4.1 Testing

The hardware units are extensively tested using an automated testing workflow to create a combination of directed and constrained random test cases for all units except TRNG. Whereby directed test cases were employed for edge cases of the units based on the algorithm and implementation (e.g., an empty string) but are of a small number, as they are manually chosen. On the contrary, the constrained random test cases were aimed to break the system by trying test cases a human would potentially fail to think about, leading to a significant number of test cases for this category. Table 4.1 demonstrates the imbalance of the number of test cases by showing the distribution of test cases applied for each IP-core.

4.1.1 Automated Testing Work-flow

The workflow uses a collection of open-source software implementations of the algorithms to ensure the generated reference result of the test cases are valid and any misunderstanding of the developer has not influenced them. It relies upon an automated script to use the software implementations to create a series of test cases for the IP-cores, which are then loaded by the hardware testbench to execute on

IP-core type	Directed Test Cases	Constrained random
AES-Encrypt	13	758
AES-Decrypt	13	826
DES	15	653
SHA-1	17	1256
SHA-2	16	1135
Montgomery Mult.	11	760

Table 4.1: Test case distribution of all IP-core

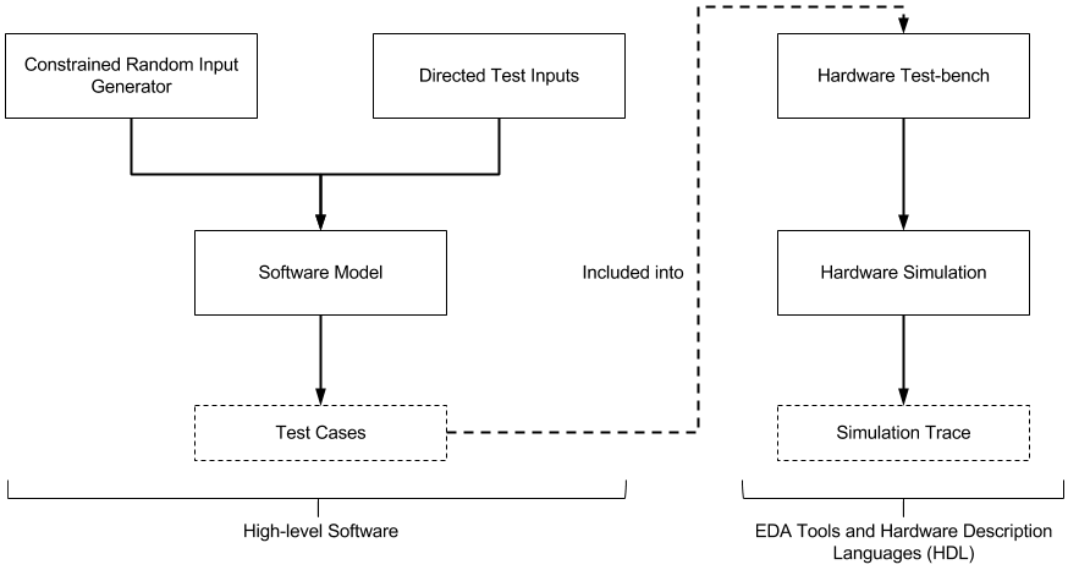


Figure 4.1: Automated Testing Work-flow

the IP-cores. The hardware testbench then has the responsibility of setting up the required hardware and verifying the results as shown in Figure 4.1.

The automated nature of the script allows for the generation of a vast number of test cases using a random input. However, the random input can potentially be of repeated test type and may not contribute towards a good coverage. Thus, the random input creation is constrained to generate valid input sequences for different kinds of input rather than being completely random, avoiding excessive repetition of test cases and ensuring that the test cases are sensible and in fact contribute towards a better test coverage. As a result, a long constrained random testing run with directed test cases have to lead to 100 % test coverage for FSM [1] states, wire toggle, expressions and blocks according to the EDA tools.

4.1.2 Testing TRNG

The testing for TRNG unit is done by testing of individual units and directed test cases to verify their connections. Followed by a manual verification of the numbers generated by the creation of very long simulation traces rather than a formal verification of the results using NIST standard due to the restriction on time available for this project. However, the verification remains valid and of confidence as the source of randomness and strong mixing property of DES have been proved earlier [69]. Further, a standalone 64-bit DES block would only repeat the output after 2^{64} combination if the input was a serial counter. When stretched to 128 bit

[1] FSM stands for Finite State Machine

IP-cores	Module	LUTs	BRAM
AES Encrypt	Structural-L	800	0
	2-stage-L	809	0
	2-stage-B	172	4
	3-stage-L	899	0
	3-stage-B	397	4
AES-Decrypt	Structural-L	932	0
	2-stage-L	1079	0
	2-stage-B	630	4
DES	Structural	96	0
SHA1	2-stage	235	0
	3-stage	295	0
SHA2	3-stage	365	0
128-bit Montgomery Mult.	FSM_Datapath	1014	0
TRNG	Structural	544	0

Table 4.2: Area requirements of all IP-core

the whole system leads to 2^{128} possible outcomes before repeating the number with a serial counter. Note, the implementation done in this study uses a random source instead of a counter which is theoretically bound to provide a much better behaviour. When combined with verification of implementation this provides a strong ground of trust for the design.

4.2 Evaluation

The IP-cores are synthesised and evaluated for a Xilinx Zynq 7020 FPGA [72] with Xilinx Vivado 2014.3.1 [73] in this study, but the designs do not use any device specific primitives and hence are portable to any different device. The FPGA and EDA tool were selected to have the same process technology and primitives as the reference NEON used to calculate area in [9]. Further, they represent a large community of researchers as their users and are readily available for wide range public at low cost compared to other the industry alternatives (e.g., Cadence and Mentor), leading to more comparable and verifiable results.

The following subsections will highlight and discuss the performance of the units in theory and the chosen benchmark.

4.2.1 Potential Performance

The measured performance of the IP-cores concern area, latency and throughput to evaluate the designs from different perspectives.

IP-cores	Module	Latency (ns)	Frequency (MHz)	T_p
AES Encrypt	Structural	7.195	138.98	1482.4
	2-Stage	4.666	214.316	2286
	3-Stage	4.58	218.34	2328.93
AES Decrypt	Structural	7.983	125.266	1457.6
	2-Stage	5.63	177.6	2066.4
DES	Structural	5.378	185.942	743.768
SHA1	2-Stage	6.32	158.227	173.36
	3-Stage	5.87	170.36	186.72
SHA2	3-Stage	5.98	167.22	359.68
128-bit Montgomery Mult.	FSM-Datapath	4.995	200	192
TRNG	Structural	5.378	185.942	N/A

Table 4.3: Timing characteristics and potential performance of all IP-cores.

Area and latency

The timing characteristics of the units are taken after Place and Route rather than Synthesis, as the timing results from the Synthesis stage tends to be more optimistic about routeing delays and do not provide a realistic behaviour. The resulting area of the units are listed in the Table 4.2, while their latency and frequency are shown in Table 4.3. Note, there exist variants for some units and these are provided in the library to support application which requires making a trade-off regarding area and performance. The AES-Encrypt with three pipeline stage is shown to be the fastest implementation in the library whereas the SHA1 is the slowest among different algorithms. This is due to the structure of the algorithms. AES has been designed to be parallel and efficient in both software and hardware while SHA1 is not. Thus, SHA1 has a more serial nature with adders being the bottleneck for performance.

Furthermore, area requirements of the IP-cores use only LUTs and BRAM for compatibility with any other FPGA fabric. BRAM are specifically used for SBox implementations of the symmetric ciphers as it makes the area requirements more compact and implementation more secure from cache-timing attacks. The smallest unit is DES due to massive optimisation gains from mapping permutation operations to wiring.

The largest IP-core is the 128-bit Montgomery multiplier because of its FSM-Datapath implementation. However, its ability to change the multiplier bit width allows varying the area footprint as required. Further, Figure 4.2 shows the iteration interval necessary for the co-processing unit running at 650 MHz. The iteration interval required before another instruction can be issued about 3 to 4 cycle for the best units in the library. It signifies that the Interlay units operate on lower clock frequency compared to NEON by a factor of 3, although it can be improved with

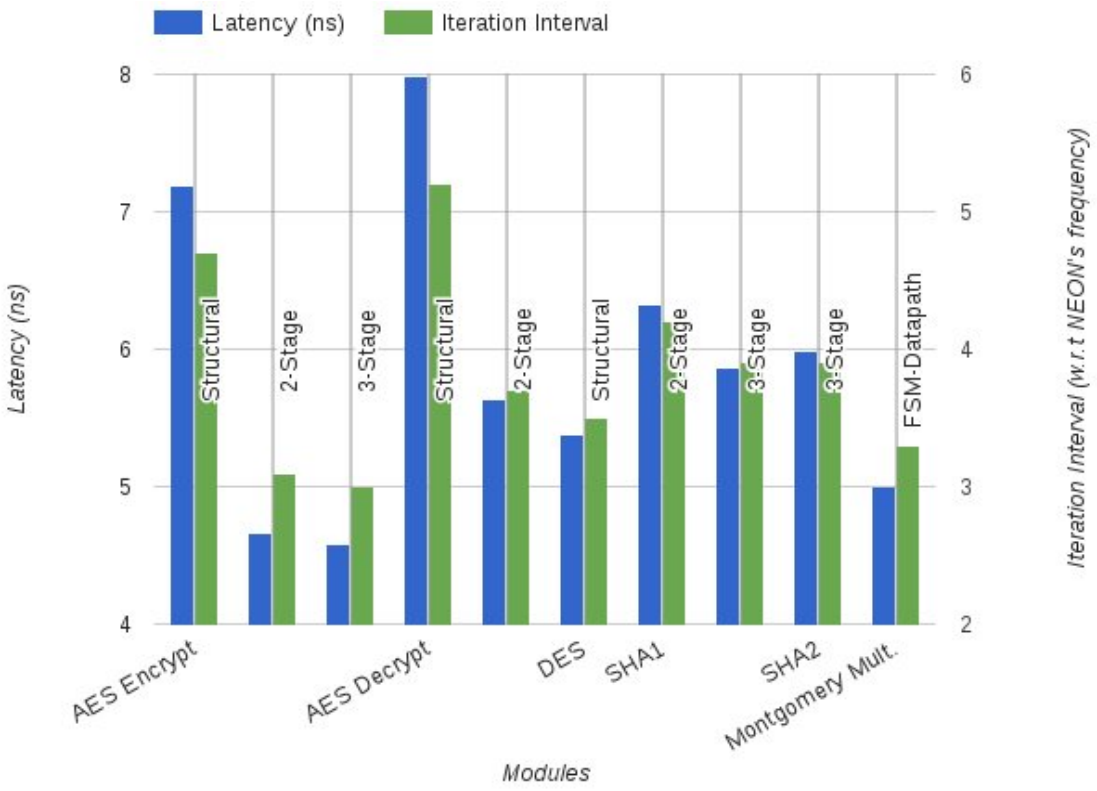


Figure 4.2: Timing from perspective of 650 MHz co-units.

certain techniques (as highlighted in Section 4.2.2). Note, higher clock frequency leads to more power consumption due to more chip activity and thus is not advisable for IoT devices.

Throughput

The peak performance of the IP-cores is defined by potential throughput (T_p) calculated from the number of cycles required (N_c), frequency (f) and bits generated for the algorithm (b) as:

$$T_p = \frac{f \times b}{N_c} \tag{4.1}$$

Note, the number of cycles used for the calculation is the cycles required when the IP-core is operating under optimal condition, i.e. the data is present for execution when necessary, and the pipeline is kept full at all times. Table 4.3 shows the result achieved.

One common trend is that increase in pipeline depth is leading to an increase in throughput, as we would expect but the gain achieved is not linear potentially due to the imbalance of logic in pipeline stages. Furthermore, usage of BRAM did

not have significant impact on the throughput or timing characteristics for the AES implementation. It is due to the operation on whole 128-bit vector concurrently using LUTs has allowed EDA tools to optimise the wiring delay and lead to lower latency compared to BRAMs as they are present at fixed locations on the FPGA. Thus being able to offset the potential gains of BRAM with wiring optimisation. Additionally, The highest throughput is achieved by AES while lowest is SHA1 owing to their timing characteristics as discussed in Section 4.2.1.

4.2.2 Benchmarking

The Section 4.2.1 discussed the potential performance under optimal conditions but to analyse the performance and behaviour under more realistic conditions, MiBench benchmark is used [74] with Gem5 simulator [75] to investigate further w.r.t. to a vector unit by ARM NEON. The comparison is meaningful and fair as the interface and area constraints of both hardware units are similar as mentioned in Section 2.1.

The investigation is done with open source Gem5 simulator because it supports a simulation of ARM Cortex-A9 CPU with NEON vector unit and allows for modification of instruction set and timing characteristics. The latter ability is used to modify the simulator to support custom instructions of IP-cores with same behaviour and model an Interlay architecture, as a real chip with an Interlay architecture does not exist yet. Further details on Gem5 settings are mentioned in Section 4.2.2.

While MiBench is preferred because it is a free commercially representative embedded benchmark suite and hence models the *real workload* of current and potentially future IoT applications. However, due to the restriction on time available for this project, only Rijndael (AES) implementation from the benchmark is used for the investigation from here on.

Gem5 setup

The Gem5 set up used in this study is as same as in [76] except the running frequency is 650 MHz instead of 800 Mz. There are two configurations of Gem5, one with and one without some modification, the first being original NEON configuration and the second with custom instructions. These custom instructions are added to simulator by taking the unused instructions of NEON for the application under examination. The timing behaviour of the added custom instruction is based on the results of IP-cores showed in the Section 4.2.1. Allowing to model the conduct of our IP-core in the Gem5 simulator. It also minimises the effort required and makes the modification safe and reliable, because no other functionality has been touched the stability and trust in the results of simulator remains intact as well.

Additionally, to use the custom configuration, MiBench is also modified to call the custom instruction using inline assembly but only for the kernel implemented by the custom unit. The remaining implementation is kept same for both custom unit and NEON.

Results

The results in Table 4.4 are from the execution of Rijndael implementation of MiBench for an input file of size 311kB with various optimisations. These optimisations include loop unrolling, function inlining, look up tables of combined SBox and shift row operations, auto-vectorisation, arrays and Fast variables. Further, the default block chaining mode in MiBench is CBC while operating on the end to end application of a single file. It stops the application of software pipelining and any other kind of operations on multiple blocks simultaneously. Leading to additional NOPs and low utilisation of the IP-core. Hence, to keep the comparison fair ECB mode is also implemented to employ software pipelining and maintain the unit busier while operating on the end to end application.

Static instruction count of the code generated shows a reduction in the number of instructions for AES encryption kernel of 69%. The throughput similarly is higher by a factor of x1.5 and x2 for CBC and ECB modes, respectively. The major delimiter in the performance has been the I/O requirements due to the file usage for both NEON and custom unit. The current best record of AES-CTR encryption is held by the implementation done in [77] for an ARM Cortext-A8 with 19.22 cycles/byte for the main encryption kernel. It is based on the bitslicing approach from [78] to provide protection against side channel attacks but requires eight independent blocks to operate. The number of cycles required per round per block is about 246 for this implementation out of which 73.4% belong to Boolean instructions, 10.8% to special permutation instructions from NEON and 15.8% to move instructions. However, if we employ the custom instructions, the number can be brought down to merely 40 cycles while requiring only three independent blocks to work upon, giving the throughput of 2.5 cycles/byte. It is because each of the 10 rounds is a custom instruction with iteration interval of 3 and load-store instruction requires 5 cycles. Further, the need of only three independent blocks allows better support for all types of block chaining mode at lower performance penalty when they are not available for example, CBC mode for a single file in MiBench.

The performance of the units can be improved further by application of the Hyper-Pipelining technique introduced by Altera [79] and overlapping the I/O with computation by using a DMA engine. Hyper-Pipelining removes the long routing delays by adding pipeline stages between logic units on the routing paths using

Gem5 Unit	NEON CBC	Custom CBC	Custom ECB	Hyper-pipelined ECB
Simulation Sec	0.032417	0.021959	0.017328	0.014103
Committed Instructions	31,189,121	15,382,828	10,185,484	8,893,418
CPU NumCycles	21,077,364	14,277,408	11,266,725	9,169,44
Simulation Instructions	31,189,121	15,382,828	10,185,484	8893418
Function calls	214,962	214,962	175,979	166,235
CPI	0.675792	0.928139	1.106155	1.031093
IPC	1.479745	1.077424	0.904032	0.969844
Throughput (MBps)	76.75	113.30	143.58	176.416
Idle Cycles	599,500	712,812	574,188	495,697

Table 4.4: Gem5 performance results.

Hyper-Registers. It would allow the units to run at potentially at CPU frequency, hence leading to more throughput in the presence of independent work. However, it would also require a significantly large register file to operate on the data without spilling to memory. The more modern architecture like ARM 64-bit have sufficient amount of registers and can help with this problem by allowing the application of software pipelining to benefit from the Hyper-pipelining.

This study models the effects of such architecture using the Gem5 simulator by adding more pipeline stage for the timing behaviour of the initial ECB design. Table 4.4 records the results of this experiment. Improvement in performance can be seen but not as linear as we would expect. This is because about one-third of the instruction in the benchmark belong to data movements alone and cannot be avoided by the use of custom instruction for encryption, hence limiting the performance acceleration possible. Thus a DMA engine is required to keep the IP-core more occupied with work and to improve the performance of I/O bound applications.

4.3 Suggestions for Interlay Fabric & NEON

A few common trend and operations have been observed to contribute towards better performance and have included as suggestions for implementation of Interlay fabric and NEON.

4.3.1 Interlay fabric

A potential Interlay fabric can implement a switch matrix for routing wires at bus widths of 8 bits for more efficiency because many cryptography algorithms, (and other applications) often work at byte level rather than individual bits. Further, the fabric should focus on LUT6, LUT4, Carry4 and BRAM FPGA logic primitives as they are common base logic requirements in IP-cores and reduce LUT2 and another

type of primitives as they are rarely used (and at times avoided completely) to reduce the fabric size further. Moreover, the BRAM blocks are read-only and heavily under utilised by a factor of 8x. Thus it is possible to use fractional BRAM instead to access each byte independently at negligible cost.

Hyper-Pipelining with a large register file can also improve performance as shown in the Section 4.2.2, and Interlays should employ it to increase the clock frequency of the IP-cores. An increase in frequency could potentially also allow the IP-core to run at CPU frequency, improving the synchronisation and communication capabilities between the Interlay and CPU.

Furthermore, a few Digital Signal processing (DSP) blocks on the Interlay can help enhance certain operations (e.g., multiplication) and serve as an alternative when other resources are consumed depending on the application.

4.3.2 NEON

The NEON vector unit already supports permutation instruction which helps accelerate the permutation operations for all the ciphers. However, without application specific techniques there is no support for substitution operation which ends up being the bottleneck for many cryptography applications. The implementation of a look-up table is not expensive when implemented in hardened logic and as results of this study shows it can improve performance and security of many cryptography applications. Further, a look-up table instruction can also be used to implement other complex computations depending upon the application. It allows for the acceleration of many different types of applications other than cryptography.

Chapter 5

Conclusions

This chapter presents the summary of the project followed by future work direction and reflection on the project experience.

5.1 Summary

The project has successfully shown that FPGA Interlay could provide better acceleration in the form of ISE than NEON vector using the same die area. It did so by highlighting the design trade-offs required for the development of IP-cores in a resource constrained environment and their respective performance and area impact concerning ideal and real-world scenarios. It has also shown how the validation of the IP-cores produced could be performed using automated testing and how a software library can be used to program them efficiently.

Furthermore, the results achieved imply that the FPGA Interlay approach can potentially solve the problem of power wall and end of Moore's law while using CMOS technology. Thus, it has clearly achieved its aims and objectives.

The artifacts produced by this project include:

- Interlay ISE implementations for all the main cryptography primitives.
- A software library for Interlay ISEs.
- Automated testing workflow.

5.2 Future Work

Interlays interface directly with a CPU or cache. It allows for very tightly coupled integration with a CPU and the software. For example, Interlays allow calling a custom instruction directly from user mode without the need of special drivers. Further, the reconfigurable nature would permit the use of custom data types, boosting the performance and lower the power consumption of applications which can operate with non-standard data types (e.g., as sometimes used for machine learning applications). It also opens a window for the combination of RISC and CISC architectures [80, 81] through a run-time reconfigurable ISE.

These possible behaviours of data types and a mix of CISC-RISC architecture have not been studied intensively before in the context of accelerating a hardened

CPU core. Moreover, it also represents challenges regarding hardware manufacturing with VLSI technology and the design and layout of the FPGA fabric to be used.

An efficient use of these characteristics and implementation would require an entire ecosystem, to develop and deploy the custom instructions, virtualise their use, and to design an Interlay fabric and its System on Chip (SoC) integration. Hence, there are many potential directions to work towards for a complete development of an FPGA Interlay architecture.

5.3 Reflection

The Interlay approach is quite new and unconventional and combined with a new set of development tools, simulators and in-line assembly, presented quite a steep learning curve. Further, the lack of work on FPGA implementations of cryptography for similar resource constraints in literature, lead to significant amount of time into research and experimentation.

Moreover, the initial objectives of the project did not involve the implementation of Montgomery multiplication and TRNG implementation, but later as the project moved forward, they were considered to be vital for completeness of the library and were included at the end of the project. However, this change was easily integrated into time span available as initial plan did assume the possibility of extensions and faults and had allocated extra time for it.

Overall the project has presented many challenges concerning domain knowledge and personal capabilities but has been a worthwhile experience. Some of the personal achievements are as follows:

- Comprehending of over thirty research papers and other material on cryptography without any security background or support.
- Writing a research paper for FPL 2017 conference.
- Modification and use of open source simulators and software.
- Conduct of benchmarking experiment.
- Automating a hybrid testing harness with open source software and EDA tools.
- Use of inline assembly and cross compilers for writing software library.

Bibliography

[1] C. A. Mack, "Fifty years of moore's law," *IEEE Transactions on Semiconductor Manufacturing*, vol. 24, no. 2, pp. 202–207, May 2011.

[2] D. Geer, "Chip makers turn to multicore processors," *Computer*, vol. 38, no. 5, pp. 11–13, May 2005.

[3] A. Peleg and U. Weiser, "Mmx technology extension to the intel architecture," *IEEE Micro*, vol. 16, no. 4, pp. 42–50, Aug. 1996. [Online]. Available: http://dx.doi.org/10.1109/40.526924

[4] ARM, "Introducing neon, development article," 2009, accessed: 2017-03-04. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dht0002a/index.html

[5] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, Oct 1974.

[6] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 365–376. [Online]. Available: http://doi.acm.org/10.1145/2000064.2000108

[7] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, and T. Razafindralambo, "A survey on facilities for experimental internet of things research," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 58–67, 2011.

[8] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 2, pp. 203–215, 2007.

[9] J. R. G. Ordaz and D. Koch, "soft-neon: A study on replacing the neon engine of an arm soc with a reconfigurable fabric," in *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, July 2016, pp. 229–230.

[10] ARM, "Arm cortex-a9 mpcore macrocell," 2014, june. [Online]. Available: www.arm.com/images/A9-osprey-hres.jpg

[11] J. Viega, M. Messier, and P. Chandra, *Network Security with OpenSSL: Cryptography for Secure Communications.* " O'Reilly Media, Inc.", 2002.

[12] N. Fact Sheet, "Suite b cryptography. national security agency," 2011.

[13] N. Sharma and V. S. Rathore, "Different data encryption methods used in secure auto teller machine transactions," *International Journal of Engineering and Advanced Technology (IJEAT) ISSN*, pp. 2249–8958, 2012.

[14] C. E. Shannon, "Communication theory of secrecy systems," *Bell Labs Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.

[15] V. Rijmen and J. Daemen, "Advanced encryption standard," *Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology*, pp. 19–22, 2001.

[16] W. C. Barker, E. Barker, U. D. of Commerce, N. I. of Standards, and Technology, *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher: NIST Special Publication 800-67, Revision 2.* USA: CreateSpace Independent Publishing Platform, 2012.

[17] D. E. Standard, "National institute of standards and technology," *Federal Information Processing Standard (FIPS) Publication*, pp. 46–1, 2002.

[18] N. FIPS, "186 digital signature standard," 1994.

[19] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978. [Online]. Available: http://doi.acm.org/10.1145/359340.359342

[20] N. Koblitz, A. Menezes, and S. Vanstone, *The State of Elliptic Curve Cryptography.* Boston, MA: Springer US, 2000, pp. 103–123. [Online]. Available: http://dx.doi.org/10.1007/978-1-4757-6856-5_5

[21] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[22] P. FIPS, "180-2," *FIPS Publication–Secure hash standard (+ Change Notice to include SHA-224)*, 2002.

[23] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," DTIC Document, Tech. Rep., 2001.

[24] S. Bellovin and M. Blaze, "Cryptographic modes of operation for the internet," in *Second NIST Workshop on Modes of Operation.* Citeseer, 2001.

[25] H. Lipmaa, D. Wagner, and P. Rogaway, "Comments to nist concerning aes modes of operation: Ctr-mode encryption," 2000.

[26] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee, "Chimaera: a high-performance architecture with a tightly-coupled reconfigurable functional unit," in *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2. ACM, 2000, pp. 225–235.

[27] L. Wei, Z. Xiaoyang, N. Longmei, C. Tao, and D. Zibin, "A reconfigurable block cryptographic processor based on vliw architecture," *China Communications*, vol. 13, no. 1, pp. 91–99, 2016.

[28] S. Sundararajana, U. Meyer-Baese, and G. Botella, "Custom instruction for nios ii processor fft implementation for image processing," in *SPIE Commercial+ Scientific Sensing and Imaging*. International Society for Optics and Photonics, 2016, pp. 987 105–987 105.

[29] P. Yiannacouras, J. G. Steffan, and J. Rose, "Vespa: portable, scalable, and flexible fpga-based vector processors," in *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*. ACM, 2008, pp. 61–70.

[30] M. Naylor, P. J. Fox, A. T. Markettos, and S. W. Moore, "Managing the fpga memory wall: Custom computing or vector processing?" in *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*. IEEE, 2013, pp. 1–6.

[31] C. H. Chou, A. Severance, A. D. Brant, Z. Liu, S. Sant, and G. G. Lemieux, "Vegas: Soft vector processor with scratchpad memory," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2011, pp. 15–24.

[32] L. Verdoscia and R. Giorgi, "A data-flow soft-core processor for accelerating scientific calculation on fpgas," *Mathematical Problems in Engineering*, vol. 2016, 2016.

[33] T. N. Prabakar, B. Lakshmi, and G. Seetharaman, "Fpga implementation of non-linear cryptography," *Circuits and Systems*, vol. 7, no. 08, p. 1250, 2016.

[34] S. Moslehpour, K. Jenab, and E. Siliveri, "Design and implementation of nios ii system for audio application," *International Journal of Engineering and Technology*, vol. 5, no. 5, p. 627, 2013.

[35] J. Yu, G. Lemieux, and C. Eagleston, "Vector processing as a soft-core cpu accelerator," in *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*. ACM, 2008, pp. 222–232.

[36] J. Cong, Y. Fan, G. Han, and Z. Zhang, "Application-specific instruction generation for configurable processor architectures," in *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*. ACM, 2004, pp. 183–189.

[37] N. Clark, W. Tang, and S. Mahlke, "Automatically generating custom instruction set extensions," *Ann Arbor*, vol. 1001, pp. 48 109–2122, 2002.

[38] O. Almer, R. Bennett, I. Böhm, A. Murray, X. Qu, M. Zuluaga, B. Franke, and N. Topham, "An end-to-end design flow for automated instruction set extension and complex instruction selection based on gcc," in *Proceedings of the First International Workshop on GCC Research Opportunities (GROW09)*, 2009, pp. 49–60.

[39] A. Bracy, P. Prahlad, and A. Roth, "Dataflow mini-graphs: Amplifying superscalar capacity and bandwidth," in *Microarchitecture, 2004. MICRO-37 2004. 37th International Symposium on.* IEEE, 2004, pp. 18–29.

[40] S. Yehia and O. Temam, "From sequences of dependent instructions to functions: An approach for improving performance without ilp or speculation," in *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on.* IEEE, 2004, pp. 238–249.

[41] P. Yiannacouras, J. G. Steffan, and J. Rose, "Exploration and customization of fpga-based soft processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 266–277, 2007.

[42] D. Koch, F. Hannig, and D. Ziener, "Fpgas for software programmers," 2016.

[43] J. Zambreno, D. Nguyen, and A. Choudhary, "Exploring area/delay tradeoffs in an aes fpga implementation," in *International Conference on Field Programmable Logic and Applications.* Springer, 2004, pp. 575–585.

[44] S. Qu, G. Shou, Y. Hu, Z. Guo, and Z. Qian, "High throughput, pipelined implementation of aes on fpga," in *Information Engineering and Electronic Commerce, 2009. IEEC'09. International Symposium on.* IEEE, 2009, pp. 542–545.

[45] T. Good and M. Benaissa, "Very small fpga application-specific instruction processor for aes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 7, pp. 1477–1486, 2006.

[46] P. Chodowiec and K. Gaj, "Very compact fpga implementation of the aes algorithm," in *International Workshop on Cryptographic Hardware and Embedded Systems.* Springer, 2003, pp. 319–333.

[47] T. Good and M. Benaissa, "Aes on fpga from the fastest to the smallest," in *International Workshop on Cryptographic Hardware and Embedded Systems.* Springer, 2005, pp. 427–440.

[48] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis, "Improving sha-2 hardware implementations," in *International Workshop on Cryptographic Hardware and Embedded Systems.* Springer, 2006, pp. 298–310.

[49] W. Sun, H. Guo, H. He, and Z. Dai, "Design and optimized implementation of the sha-2 (256, 384, 512) hash algorithms," in *ASIC, 2007. ASICON'07. 7th International Conference on.* IEEE, 2007, pp. 858–861.

[50] R. P. McEvoy, F. M. Crowe, C. C. Murphy, and W. P. Marnane, "Optimisation of the sha-2 family of hash functions on fpgas," in *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on.* IEEE, 2006, pp. 6–pp.

[51] R. García, I. Algredo-Badillo, M. Morales-Sandoval, C. Feregrino-Uribe, and R. Cumplido, "A compact fpga-based processor for the secure hash algorithm sha-256," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 194–202, 2014.

[52] K. Javeed, X. Wang, and M. Scott, "Serial and parallel interleaved modular multipliers on fpga platform," in *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*. IEEE, 2015, pp. 1–4.

[53] K. Javeed and X. Wang, "Radix-4 and radix-8 booth encoded interleaved modular multipliers over general f p," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*. IEEE, 2014, pp. 1–6.

[54] A. M. AbdelFattah, A. M. B. El-Din, and H. M. Fahmy, "An efficient architecture for interleaved modular multiplication," *World Academy of Science, Engineering and Technology*, vol. 56, 2009.

[55] C. K. Koc, T. Acar, and B. S. Kaliski, "Analyzing and comparing montgomery multiplication algorithms," *IEEE micro*, vol. 16, no. 3, pp. 26–33, 1996.

[56] K. Shigemoto, K. Kawakami, and K. Nakano, "Accelerating montgomery modulo multiplication for redundant radix-64k number system on the fpga using dual-port block rams," in *Embedded and Ubiquitous Computing, 2008. EUC'08. IEEE/IFIP International Conference on*, vol. 1. IEEE, 2008, pp. 44–51.

[57] K. Kelley and D. Harris, "Very high radix scalable montgomery multipliers," in *System-on-Chip for Real-Time Applications, 2005. Proceedings. Fifth International Workshop on*. IEEE, 2005, pp. 400–404.

[58] K. Javeed, D. Irwin, and X. Wang, "Design and performance comparison of modular multipliers implemented on fpga platform," in *International Conference on Cloud Computing and Security*. Springer, 2016, pp. 251–260.

[59] K. H. Tsoi, K. Leung, and P. H. W. Leong, "Compact fpga-based true and pseudo random number generators," in *Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on*. IEEE, 2003, pp. 51–61.

[60] C. Petit, F.-X. Standaert, O. Pereira, T. G. Malkin, and M. Yung, "A block cipher based pseudo random number generator secure against side-channel key recovery," in *Proceedings of the 2008 ACM symposium on Information, computer and communications security*. ACM, 2008, pp. 56–65.

[61] Y. Mao, L. Cao, and W. Liu, "Design and fpga implementation of a pseudo-random bit sequence generator using spatiotemporal chaos," in *Communications, Circuits and Systems Proceedings, 2006 International Conference on*, vol. 3. IEEE, 2006, pp. 2114–2118.

[62] W. Diffie and M. E. Hellman, "Special feature exhaustive cryptanalysis of the nbs data encryption standard," *Computer*, vol. 10, no. 6, pp. 74–84, 1977.

[63] R. C. Merkle, "One way hash functions and des," in *Conference on the Theory and Application of Cryptology*. Springer, 1989, pp. 428–446.

[64] B. Hatch, J. Lee, and G. Kurtz, *Hacking Linux exposed: Linux security secrets & solutions*. Osborne/McGraw-Hill New York, 2001.

[65] V. Bunimov, M. Schimmler, and B. Tolg, "A complexity-effective version of montgomerys algorithm," in *Workshop on Complexity Effective Designs, ISCA*, vol. 2. Citeseer, 2002.

[66] S. H. Standard, "Fips pub 180-2," *National Institute of Standards and Technology*, 2002.

[67] S. S. Keller, "Nist-recommended random number generator based on ansi x9. 31 appendix a. 2.4 using the 3-key triple des and aes algorithms," *NIST Information Technology Laboratory-Computer Security Division, National Institute of Standards and Technology*, 2005.

[68] Y. Ma, J. Lin, and J. Jing, "On the entropy of oscillator-based true random number generators," in *Cryptographers Track at the RSA Conference*. Springer, 2017, pp. 165–180.

[69] D. Schellekens, B. Preneel, and I. Verbauwhede, "Fpga vendor agnostic true random number generator," in *2006 International Conference on Field Programmable Logic and Applications*, Aug 2006, pp. 1–6.

[70] J. Kalisz and Z. Jachna, "Metastability tests of flip–flops in programmable digital circuits," *Microelectronics Journal*, vol. 37, no. 2, pp. 174–180, 2006.

[71] L.-S. Kim and R. W. Dutton, "Metastability of cmos latch/flip-flop," *IEEE Journal of solid-state circuits*, vol. 25, no. 4, pp. 942–951, 1990.

[72] Xilinx, "Zynq-7000 all programmable soc technical reference manual," 2016, accessed: 2017-03-08. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf

[73] Xilinx, "Vivado design suite user guide," accessed: 2017-03-08. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_3/ug973-vivado-release-notes-install-license.pdf

[74] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*. IEEE, 2001, pp. 3–14.

[75] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[76] F. A. Endo, D. Couroussé, and H.-P. Charles, "Micro-architectural simulation of in-order and out-of-order arm microprocessors with gem5," in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on*. IEEE, 2014, pp. 266–273.

[77] D. J. Bernstein and P. Schwabe, "Neon crypto," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2012, pp. 320–339.

[78] E. Käsper and P. Schwabe, "Faster and timing-attack resistant aes-gcm," in *Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer, 2009, pp. 1–17.

[79] Altera, "Hyper-pipelining for stratix 10 designs," 2015, accessed: 2017-03-14. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/an/an715.pdf

[80] D. A. Patterson and D. R. Ditzel, "The case for the reduced instruction set computer," *ACM SIGARCH Computer Architecture News*, vol. 8, no. 6, pp. 25–33, 1980.

[81] S. C. J. Garth, "Combining risc and cisc in pc systems," in *IEE Colloquium on RISC Architectures and Applications*, Nov 1991, pp. 10/1–10/5.