FLEXIBLE AND RESOURCE EFFICIENT DESIGN FOR HARDWARE IMPLEMENTATION OF THE ADVANCED ENCRYPTION STANDARD

CHENG WANG

001311

Flexible and Resource Efficient Design for Hardware Implementation of the Advanced Encryption Standard

by

© Cheng Wang, M.Sc., B.Eng.

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the

requirements for the degree of

Doctor of Philosophy

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

March 2012

St. John's Newfoundland Canada

Abstract

In this dissertation, we investigate the performance of a broad range of hardware implementations of symmetric key block ciphers. The major focus of this dissertation is dedicated to the investigation of the performance of hardware implementations of the Advanced Encryption Standard (AES) influenced by the implementation architecture. The flexibility of the AES algorithm allows an extensive variety of implementation architectures making it necessary to investigate the performance characteristics of these architectures. On the one hand, this investigation identifies the most resource-efficient architecture for the AES implementation targeted at resource-constrained applications; while, on the other hand, this discloses the unique performance trade-offs from the different architectures allowing flexible implementation of AES.

In this dissertation, two perspectives of the implementation architecture of AES are explored: 1) the pipeline configuration of the AES S-box with the composite field structure and 2) the datapath architecture of AES. For the S-box pipeline configuration, a gate-level approach for the pipeline of 2 to 7 stages and a component level approach for the pipeline of 2 to 4 stages are evaluated. For the datapath architecture, parameterized architectures with the datapath width of 8, 16, 32, 64 or 128 bits and the unrolling factor of 1, 2, 5 or 10 for the 128-bit width are evaluated, among which the 16, 32 and 64-bit architectures are novel designs.

Generic and typical models for these architectures are built. The performance of these implementations in terms of timing, area, power and energy is benchmarked based on Complementary Metal-Oxide Semiconductor (CMOS) technology following Application-Specific Integrated Circuit (ASIC) design flow. The benchmark results demonstrate the potentially significant performance improvement by selecting an appropriate architecture under given throughput requirements compared with other architectures.

For the investigation of S-box pipeline configurations, we examine the performance of the S-box implementation for the 9 configurations over a wide range of throughput requirements. Based on the results, we analyze the influence of the pipeline configuration on the performance and identified the regular trends of the resource costs varying with the pipeline configurations and the throughput requirement/timing constraint. Numerically, there are maximum reductions of 51% in area and 69% in power/energy with an appropriate pipeline configuration throughput requirement/timing constraint comparing with other configurations including no pipeline. For the investigation of datapath architecture, we examined the performance of the datapath implementation for the 8 datapath architectures over a wide range of throughput requirements. The quantitive performance of the architecture implementations is presented, compared and analyzed. The most efficient architecture implementation in area, peak power and energy, as well as in the overall resource cost is identified. The performance trade-offs over a range of architecture parameter values are disclosed. In contrast to conventional belief, the most compact architecture implementation with the 8-bit width does not help to minimize, but actually increases, the energy consumption even running at a low clock frequency. As well, compact implementations do not result in the minimal peak power. In our research, we also combine the most energy efficient S-box pipeline configuration and datapath architecture identified from the above and demonstrate the further reduction in energy consumption by the combination. It is found that the combination consumes about 50% less energy than solely the most energy efficient datapath architecture implementation and 80% less energy than the 8-bit width datapath architecture implementation.

The last chapter of this dissertation discusses the design of a newly proposed block cipher named PUFFIN2. PUFFIN2 is designated for lightweight applications with modest security strength requirements. PUFFIN2 has an involutional structure and can lead to a very compact implementation, which is smaller than the previous most compact block cipher PRESENT.

Acknowledgements

I would like to deeply express my sincere appreciation and respect to my supervisor Dr. Howard M. Heys. Being his student is really a beneficial and pleasurable experience. His wisdom and knowledge guided me through the difficulties in my studies and research. His kindness and tolerance added lots of flexibility and comfort to my life during the studies and research. His rigorous scholarship and justice convinced me to follow him for lifetime.

I also need to direct my thanks to my supervisory committee members Dr. Ramachandran Venkatesan and Dr. Lihong Zhang, who made efforts to support me, review my thesis and give valuable suggestions. I would also like to thank Dr. Cheng Li and Dr. Theodore S. Norvell for their helps.

It is my great joy to know and get along with many good friends and lab mates in the university, including Dr. Yuanlong Yu, Xiaoning Zhang, Jiankang Wang, Amir Zadeh, Ruoyu Su and Zehua Wang. Thanks for their accompany, support and encouragement. Special gratitude to Dr. Yuanlong Yu for listening to me, understanding me and encouraging me all along the days we work towards Ph.D. degree.

I am grateful to my landlords Betty and Steve, with whom I live since the first week I came here. They accommodate me with a comfortable and carefree home so that I can concentrate on my work at the university. They leave a light on every night for my late getting home, which makes me feel warm even in the coldest days.

Finally, I would like to mention that there is no way to fully appreciate my parents for their love and support and there is also no way to erase my regret not able to accompany them over the years as the only child of them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In 2002, the Advanced Encryption Standard (AES) was established as the symmetric key encryption algorithm that is officially endorsed by the United States government for the next decades [1]. Since then, this algorithm has been widely adopted by applications requiring security features. This has led to intensive research and development of the efficient hardware implementation of AES. Although AES was introduced around a decade ago and the most recent effort on the cryptanalysis has been able to reduce the computational complexity for a key recovery by a factor of 1/4 compared with a brute force attack [2], its security is still regarded to be sufficient for future needs.

Since the information security provided by AES is often realized through the encryption and/or decryption of data transmitted over communication channels, a basic requirement for a hardware implementation of AES is the throughput. Upon meeting this basic requirement, the resource cost of the implementation is expected to be as low as possible. The resource cost of an implementation includes the manufacture cost and the running cost. The manufacture cost is related to the complexity of the circuit, which is usually measured by the area of the circuit. The running cost usually denotes the power consumption and/or energy consumption when running the implementation. Power consumption and energy consumption of the implementation are significant metrics for passively powered devices (e.g., contactless smart cards and RFID tags) and battery powered devices since the battery life and the potential workload are determined by these consumptions.

1.1 Motivation

Due to the round-iterative nature of the AES algorithm, as well as the highly regular and repeating operations defined in the datapath, the design of the datapath architecture for the hardware implementation of the algorithm is extremely flexible and, as a result, there exists a number of possible datapath architectures. However, in the previous literature, only a small part of them have been discussed and investigated. In addition, the AES S-box, as the most complex component in an AES implementation, suffers from a long critical path and is conventionally pipelined only for the benefit of speedup. However, there exists more pipeline architectures than those adopted in previous works and also unexplored potential benefit for resource efficiency when using pipelining.

Since each of these architectures (including both the datapath architectures and the S-box pipeline architectures) could lead to the implementation with unique performance characteristics and performance trade-offs, it is necessary to conduct an investigation of them for the comprehensive understanding of the performance variation under the different architectures, especially with the focus on the resource-related performance. On one hand, this would identify the most resource-efficient architecture for the AES implementation targeted at resource-constrained applications. On the other hand, the performance trade-offs from these architectures would provide more options for the flexible implementation of AES.

The design of the block cipher PUFFIN2 is motivated by the demand for block ciphers targeted at lightweight applications. Since AES has relatively high implementation complexity as is determined by the algorithm, there is demand for an alternative block cipher algorithm that leads to lower implementation complexity suitable for low price and low power applications. For these applications, the block cipher is not required to be as strong as AES in terms of security level (i.e. the effort required to crack the cipher) but the implementation complexity is expected to be as low as possible.

1.2 Dissertation Outline

The dissertation is outlined as the follows. Chapter 2 is the review of the AES algorithm and the previous works on the hardware implementation of AES. Through the analysis of the previous work, the necessity of the work presented in this dissertation is shown. In Chapter 3, the methodology of the hardware implementation and the performance evaluation adopted in this dissertation is introduced. Chapter 4 is a preliminary study of using pipelined AES S-boxes for resource efficient purpose. In this work, we replace the two S-boxes in an ultra compact AES implementation with one pipelined S-box and are therefore able to double the throughput while the area of the implementation remains very similar. Inspired by the work in Chapter 4, we conduct a comprehensive study of the performance of the pipelined AES S-box with various pipeline configurations in Chapter 5. This work shows a more complete picture of the value in using pipelined S-boxes over non-pipelined in terms of resource efficiency and design flexibility. Based on the results in this chapter, we also analyze the generalizable trend of the variation of the appropriate pipeline configuration over the different throughput requirements. In Chapter 6, we investigate the resource efficiency and performance trade-offs of the datapath implementation of AES with various datapath architecture and throughput requirements. In contrast to conventional belief, this investigation discloses that the most power and energy efficient datapath architectures under a given throughput requirement are not achieved by the most compact architecture and there is significant reduction in power and energy by using other appropriate datapath architectures. In Chapter 7, we demonstrate the combined effectiveness in improving resource efficiency of the results from Chapters 5 and 6 by examining the performance of the implementation with the combination of a S-box pipeline configuration and a datapath architecture. In Chapter 8, we present the design of PUFFIN2 with the relevant background of lightweight block cipher design. Chapter 9 is the summary of the research and contributions in this dissertation and the suggestions for future work.

Chapter 2

AES and the Hardware Implementations

In this chapter, the algorithm of AES is introduced. Previous work on high performance and resource efficient AES hardware implementation is reviewed. The necessity of the research presented in this dissertation is shown based on the review.

2.1 The AES Algorithm

AES is a block cipher algorithm with a block size of 128 bits. The key size of AES can be independently specified to 128, 192 or 256 bits, and accordingly there are 10, 12 or 14 iterative rounds to be performed for the encryption or decryption of a block of the plaintext or ciphertext [1]. In the next section, a 128-bit block size and 128-bit key size are used for demonstration. The 128-bit plaintext block, key, round keys and the intermediate results (called the *State*) between operations in AES are organized in a rectangle array of bytes, as shown in Figure 2.1. Such an arrangement of bytes

Byte 0,0	Byte 0,1	Byte 0,2	Byte 0,3
Byte 1,0	Byte 1,1	Byte 1,2	Byte 1,3
Byte 2,0	Byte 2,1	Byte 2,2	Byte 2,3
Byte 3,0	Byte 3,1	Byte 3,2	Byte 3,3

Figure 2.1: Layout of 128-bit plaintext/*State* block, key and state

facilitates some of the operations of AES which work on rows or columns of an array.

2.1.1 Round Function

A round function of AES can be described in pseudo code notation as:

```
Round(State,Round Key)
  {
    SubBytes(State);
    ShiftRows(State);
    MixColumns(State);
    AddRoundKey(State,Round Key);
  }.
```

All round functions are the same in AES except the final round which is slightly different and expressed as:

```
FinalRound(State,Round Key)
  {
    SubBytes(State);
    ShiftRows(State);
    AddRoundKey(State,Round Key);
  }.
```

In the notation above, *State* denotes the intermediate result produced by the preceding operation. *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey* are operations in round functions and they are defined in the next section.

The *SubBytes* operation works as the substitution layer of AES. The 8-bit non-linear transformation of *SubBytes* is referred to as the S-box. The operation performs the non-linear transformation of each byte in the *State* according to the S-box mapping of AES. The S-box mapping is the composition of two transformations: (1) the multiplicative inverse over $GF(2^8)$ with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ (with the exception of mapping zero to zero) and (2) an affine transformation defined by

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{2.1}$$

where x_n and y_n are the $(n+1)$-th bit of the input byte and output byte, respectively, of the affine transformation.

The *ShiftRows* operation cyclically shifts left the bytes in the rows of the *State* with different offsets as illustrated in Figure 2.2.

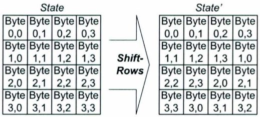


Figure 2.2: The *ShiftRows* operation of AES

In the *MixColumns* operation, the columns of the *State* are considered as polynomials with coefficients in $GF(2^8)$ and multiplied modulo $m(x) = x^4 + 1$ with a fixed polynomial $c(x) = 03x^4 + 01x^2 + 01x + 02$. Given the $(n+1)$-th column, $n \in \{0, 1, 2, 3\}$, of the *State* $B_{0,n}$, $B_{1,n}$, $B_{2,n}$ and $B_{3,n}$, $(n+1)$-th column of the *State* after *MixColumns* can be realized by the matrix multiplication as

$$\begin{bmatrix} B'_{0,n} \\ B'_{1,n} \\ B'_{2,n} \\ B'_{3,n} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} B_{0,n} \\ B_{1,n} \\ B_{2,n} \\ B_{3,n} \end{bmatrix}. \tag{2.2}$$

The *ShiftRows* operation and *MixColumns* operation can be viewed as a linear transformation on the *State*.

The *AddRoundKey* operation performs the bitwise XOR of the *State* and the round key. Each byte of the *State* is XORed with the byte of the round key with the same position in the array.

2.1.2 Datapath

An AES cipher with 128-bit block size and 128-bit key size has ten rounds in the datapath. The datapath for encryption is described in pseudo code notation as:

```
$\boldsymbol{Datapath}(Plaintext, Round\ Key)$
  {
    $\boldsymbol{AddRoundKey}(Plaintext, Round\ Key[1]);$
    $for\ (i = 1; i < 10; i++)$
      $\boldsymbol{Round}(State, Round\ Key[i+1]);$
    $\boldsymbol{FinalRound}(State, Round\ Key[11]);$
  }.
```

Note that 11 round keys are required: one prior to the first round and one for each round. The datapath for decryption is achieved by reversing the encryption datapath.

2.1.3 Key Schedule

The key schedule of AES consists of two components: (1) the key expansion where the 128-bit key is expanded into 44 32-bit vectors; (2) the round key selection where the 44 32-bit vectors are segmented into 11 128-bit vectors, each of which is a round key. The key expansion is described as follows:

```
$\boldsymbol{KeyExpansion}(byte\ Key[16], word\ W[44])$
  {
    $for\ (i = 0; i < 4; i++)$
      $W[i] = (Key[4*i], Key[4*i+1], Key[4*i+2], Key[4*i+3]);$
    $for\ (i = 4; i < Nb*(Nr+1); i++)$
      {
        $temp = W[i-1];$
        $if\ (i\%4 == 0)$
          $temp = \boldsymbol{SubBytes}(\boldsymbol{RotBytes}(temp)) \oplus Rcon[i/4];$
        $W[i] = W[i-4] \oplus temp$;
    }
}.
```

In the above description, *RotBytes* denotes a cyclic permutation operation that rotates left the bytes in the word by one byte (e.g. a word (a, b, c, d) is shifted to (b, c, d, a)) and $\oplus$ is bitwise XOR operation; *Rcon*$[i/4]$ is a pre-defined constant. The round key selection is described as:

```
SubKeySel(word $W[44]$, 4-word Round Key$[11]$)
{
    for ($i = 1; i <= 11; i++$)
        Round Key$[i] = (W[4*i], W[4*i+1], W[4*i+2], W[4*i+3])$;
}.
```

2.1.4 Modes of Operation

As a block cipher, AES needs to work under a certain block cipher mode of operation when used in applications. Basic block cipher modes of operation include Electronic Codebook (ECB) mode, Cipher Block Chaining (CBC) mode, Cipher Feedback (CFB) mode, Output Feedback (OFB) mode and Counter (CTR) mode [3]. Block cipher modes of operation can be categorized as either feedback modes (e.g., CBC, CFB, and OFB) or non-feedback modes (e.g., ECB and CTR) depending on whether each encryption/decryption using the block cipher depends on the output of the previous encryption/decryption. In this dissertation, some of the implementations are assumed to work under non-feedback modes only.

2.2 Hardware Implementation of AES

Since AES was proposed in 2001, there has been intensive investigations into the efficient hardware implementation of AES. Generally, these investigations focus on two areas: 1) the implementation of the AES S-box with low resource cost and 2) the implementation of the entire cipher or the datapath of AES for a variety of design requirements from high throughput to low resource cost. The following is the review of the S-box and datapath implementations.

2.2.1 S-Box Implementation

In the hardware implementation of an AES cipher, the S-box typically has a complexity significantly higher than other functions and therefore has a major impact on the performance of the overall AES implementation in terms of timing, area, power and energy. For this reason, various hardware designs of the S-box have been proposed aiming at the improvement of the efficiency in these perspectives of the performance.

2.2.1.1 Hardware Structures of the AES S-Box

The most straightforward realization of an AES S-box is a lookup table. A table lookup in hardware can be constructed with either a ROM or a combinational circuit. The ROM approach requires the space of 256 bytes to store the lookup table, which is costly in hardware area, particularly when multiple copies of the S-box need to be implemented in an AES implementation. The combinational circuit approach usually relies on a synthesis tool to translate the lookup table into the circuit and, due to the high nonlinearity of the lookup table, the synthesis tool has to interpret the transformation as a random mapping, resulting in a circuit with large area being generated.

To achieve better area efficiency, some advanced approaches are proposed by exploiting the mathematical properties of the S-box mapping. Since the most costly operation in the S-box is multiplicative inverse over $GF(2^8)$, Rijmen suggested in [4] to decompose the finite field $GF(2^8)$ to its sub-field $GF(2^4)$ so that an element a in $GF(2^8)$ can be represented by a polynomial $a = bx + c$ with b, c in $GF(2^4)$, and then the computation of the multiplicative inverse over $GF(2^8)$ is converted to operations over $GF(2^4)$ which have much lower complexity in hardware. An ASIC implementation of this approach is shown in [5]. This approach is further developed in [6] by decomposing the elements in $GF(2^4)$ to polynomials with coefficients in $GF(2^2)$, and in such a way the multiplicative inverse in $GF(2^2)$ can be implemented in hardware with only a swap of the two input bits. With the same decomposing approach, better area efficiency is achieved in [7] by using elaborately selected normal bases for element representation and other optimization techniques. These approaches are all based on the decomposition of the operations in $GF(2^8)$, so they are generally known as S-boxes with composite field structures. There are other S-box structures proposed based on the composite field structures, which further improve the performance in timing, area and/or power through extensive exploration of the possible construction configurations using polynomial basis (e.g., [8]), normal basis (e.g., [9], [10], [11] and [12]) or mixed basis (e.g., [13]).

The S-box implementation with low power consumption can be achieved by reducing the switching activity of the circuit. A typical switching reduction design is present in [14]. It employs a decoder-permutation-encoder structure where the decoder converts the binary representation of the 8-bit input to the one-hot representation of 256 bits and the encoder does the inverse. In this way, any transition of the S-box input can only cause the transitions of the signals on two lines in the permutation (one from 1 to 0 and the other from 0 to 1), so the switching activity is minimized in the permutation. Further, the decoder and encoder are also built with power efficiency through the extensive search of all the possible structures. A low power structure targeted at ASIC implementation of the composite field structure of the AES S-box is shown in [15]. In this work, a multi-stage two level logic structure with buffers is used to balance the signal arrival times of gate so that the power consumption caused by dynamic hazards in the circuit are reduced. Other low power designs of the composite field structure of the AES S-box include [16] and [17]. The implementation in [16] is a full-custom circuit design using pass transistor logic, which usually has better power efficiency compared with the conventional CMOS logic. The work in [17] has a 7-stage pipelined S-box with the composite field structure based on Algebraic Normal Form (ANF) representation and it achieves low power consumption by reducing the hazards in the circuit through pipelining.

2.2.1.2 Performance of the AES S-Box Structures

The performance of the S-box structures mentioned above in terms of timing, area and power are characterized based on a 0.25-μm standard cell CMOS process in [18]. According to this characterization, the composite field structures have a small range of performance in all perspectives compared with other structures under examination. By varying the synthesis constraints, the implementations of the composite field structures can be built with the critical path delays in the range between 4.93

ns to 9 ns, the areas in the range between 303 GE to 625 GE (where 1 GE or Gate Equivalent is the size of a two-input NAND with the lowest drive strength in the process library), and the normalized power consumption (with respect to 4.45 μW and measured at 50 MHz) in the range between 1.51 to 2.0. The implementations of the combinational circuit structure for a lookup table has the performance in timing, area and power in the ranges of 1.95 ns to 6.61 ns, 1301 GE to 1545 GE, and 1.00 to 1.18, respectively. The performance for the implementations of the decoder-permutation-encoder structure ranges from 1.86 ns to 3.31 ns, 1399 GE to 2016 GE, and 0.27 to 0.42, respectively. The performance of the composite field structures in [8], [9], [10], [11], [12] and [13] is not included in the above characterization, but according to the comparisons in the original papers, their performance are very close to those composite field structures examined in [18] compared with lookup table structure and the decoder-permutation-encoder structure. For the low power designs of the S-box with the composite field structures like [15], [16] and [17], according to the original papers, the power consumption of [15] is about 1/4 of [6]; the power consumption of [16] is very close to [15]; and the power consumption of [17] is about half of [15].

These characterization results from [18] reveal that the composite field structures are superior in area efficiency but inferior in timing and power consumption. On the contrary, the lookup table structure and the decoder-permutation-encoder structure have the performance with the reverse trade-offs. Thus, they are usually targeted at applications with different design requirements.

2.2.1.3 Pipelined AES S-Box Implementations

The straightforward effect of pipelining is the reduction of critical path delay at the cost of the hardware overhead of pipeline registers. As is mentioned above, the composite field structures lead to implementations with low complexity and high critical path delay compared with the lookup table structure and the decoder-permutation-encoder structure. The low complexity implies fewer number of pipeline registers in pipelining and the high critical path delay implies more potential in delay reduction by pipelining. Therefore, the composite field structures are more suitable for pipelining than the other structures.

Previous work using pipelined S-boxes with composite field structures are targeted at high throughput implementations with the fully unrolled architecture. Typical examples include ASIC implementations [19] [20] or FPGA implementations [21] [22]. In [20], a throughput of around 66 Gbps is achieved by the AES implementation with 3-stage pipelined S-boxes using a 0.18-μm CMOS technology. In [22], the implementation produces a throughput of around 31 Gbps with the S-boxes pipelined into roughly 7 stages on a Xilinx Spartan-III FPGA. In [21] and [22], the relation between the critical path delay and the number of cascaded Lookup Tables (LUTs) is analyzed and, based on the analysis as well as the consideration of the availability of routing resources, the minimum realistic numbers of LUT-levels of 3 and 2 are determined for one pipeline stage and accordingly the 4-stage and 7-stage pipelines are adopted to produce the maximum throughput, respectively. In [19] and [20], the trade-off between throughput and area of the overall implementation is explored by adjusting the number of pipeline stages. However, only 2 and 4 stages are considered in [19] and 2 and 3 stages in [20]. The design in [17] is another FPGA-based design applying pipelining. However, the focus of the work is the comparison between an 7-stage pipelined S-box based on the optimum construction of the composite field proposed by the author, an 5-stage pipelined S-box based on an conventional construction of the composite field and the S-box from [15]. The results show that the 7-stage pipelined S-box with the proposed construction of the composite field has the similar power consumption but much higher throughput compared with the other designs.

Pipelining the AES S-box implementation involves the decision concerning where is the appropriate placement of the pipeline registers. While this is carefully investigated towards the realistic minimum critical path delay in [22], the FPGA-specific approach is not applicable in a standard ASIC design flow. For the ASIC implementations in [19] and [20], a similar placement approach is adopted that has the pipeline registers inserted only between the components of the S-box (e.g., the multiplicative inverter and the multiplier). Since this is a coarse-grain approach, the critical path delays are not necessarily well balanced due to the unbalanced complexities in the components. As well, the limited positions for register insertion of this component level approach would prevent implementing more pipeline stages than the 4 stages in [19] and [20].

2.2.1.4 Potential for S-box Pipelining

Based on the review above, it can be seen that the AES S-box implementation for high speed, low area and low power purposes has been extensively explored. However, the pipelined S-box implementations are exclusively targeted at high throughput and only the advantage in timing or some trade-offs between timing and area are considered. In addition, the appropriate placement of the pipeline registers is not explored based on ASIC technology. For these reasons, the research presented in Chapter 5 is the investigation of the pipelined S-box implementations in terms of timing, area, power and energy and this is based on an extensive exploration of the placement of the pipeline registers targeted at ASIC technology, including an automatic placement approach for 2 to 7-stage pipelines and an manual placement approach for 2 to 4-stage pipelines.

2.2.2 Datapath Implementation

As is shown in the algorithm of AES, AES has a round iterative structure where round functions are identical (with the exception of the last round) and in each round function the *SubBytes* operation on each byte and the *MixColumns* operations on each 4-byte group are identical. This nature of AES allows for the high flexibility in the implementation architecture of AES in the way that different amounts of the parallelism of the identical operations in the implementation leads to different implementation architectures. As the general trade-off, the more parallelism in the implementation, the more throughput is achieved but the higher complexity is incurred. By exploring the trade-off, different design requirements of AES implementation can be fulfilled with different implementation architectures. The previous work on the overall implementation of AES usually focuses on the fulfillment of the design requirements of either high throughput or low resource cost.

2.2.2.1 High Throughput Implementation

The implementation with high throughput is usually fulfilled with the fully unrolled architecture with round-level pipeline, which provides the maximum level of parallelism of the identical operations and consequently the maximum throughput. Another factor that affects throughput is the critical path delay of the implementation. As the most complex component in an AES implementation, the S-box along with other components in a round can be pipelined for short critical path delay, which is known as inner round pipelining. Pipelined S-boxes/rounds are used in the implementation with the fully unrolled architecture for further improvement of the throughput.

An example of the fully unrolled architecture with round-level pipelining is [23] where a fully pipelined AES implementation achieves the throughput of 17.8 Gbps based on an Xilinx Virtex-II FPGA. There are many more works investigating inner round pipeline for maximum throughput, including ASIC implementations of [19] and [20] and FPGA implementations of [21], [24] and [22]. In [20], a throughput of around 66 Gbps is achieved by the AES implementation with 3-stage pipelined S-boxes using a 0.18-μm CMOS technology. In [24], the AES implementation has 5-stage inner round pipeline for each of the rounds and achieves a 26.64 Gbps based on a Virtex-E FPGA. In [22], the implementation produces a throughput of around 31 Gbps with the S-boxes pipelined into roughly 7 stages on a Xilinx Spartan-III FPGA.

2.2.2.2 Low Resource Cost Implementation

In contrast to the high throughput implementation, typical low resource cost implementations of AES exploit the reverse trade-off between throughput and complexity.

While the high throughput implementations of AES usually have a fully implemented datapath of 10 rounds with 128-bit width, lower complexity of the implementation can be achieved by reducing the number of rounds implemented and the datapath width based on a loop structure. While the reducing of the complexity, the throughput is also compromised since the parallelism in processing of data is reduced.

Depending on the design requirement for throughput and complexity, many AES implementations have been proposed based on a loop structure with various datapath widths. Among those mostly seen are the implementations with one round loop and the datapath widths of 128-bit, 32-bit or 8-bit. A typical example of the 128-bit width implementation is shown in [25]. For the lower complexity AES implementation, there are 32-bit AES implementations that reuse the datapath for 4 times for the operation of one round, as is seen in [26], [27], [28] and [29]. By breaking the 32-bit operation of $MixColumns$ into serialized operation on 8-bit data, some recent designs realize the AES implementation with a 8-bit datapath width for even lower implementation complexity. Designs with an 8-bit width datapath include [30], [31] and [32]. The 8-bit AES implementations are also referred as the implementations with a fully serialized architecture.

Regarding the performance of these low resource cost implementations, there is usually no uniform benchmark for the comparison between them in the way that they are usually implemented based on different platforms (ASIC or FPGA) with different technologies or devices and the performance variation caused by difference in implementation technology is not ignorable. It is also necessary to point out that these implementations with the fully serialized architecture are targeted at the design requirement of low resource cost, which means low in area, power and energy.

However, while it is straightforward to see the achievement of low area and that power consumption can be scaled down by decreasing the clock frequency, it is not clear if these implementations lead to low energy cost. The detailed distinction between the power consumption and the energy consumption of an implementation is introduced in Section 3.2.2.

2.2.2.3 Potential for Different Architectures

According to the above review, most of the previous work focuses on a specific architecture for a specific design requirement, usually either the high throughput implementations with the fully unrolled datapath architecture or the low area implementations with loop based 128-bit, 32-bit or 8-bit width datapath. However, there has been insufficient investigation on the performance of other datapath architectures as well as the evaluation of the performance of the various datapath architectures based on the same benchmark, especially in terms of power and energy. In Chapter 6, we conduct the investigation of an extensive range of the datapath architectures for their performance in timing, area, power and energy based on a 90-nm standard cell ASIC technology. These architectures include parameterized architectures with the datapath width of 8, 16, 32, 64 or 128 bits and the unrolling factor of 1, 2, 5 or 10 for the 128-bit width. Through this investigation, the performance trade-offs between timing, area, power and energy over the different architectures are shown and the power and energy efficient architectures are identified.

2.2.3 Characterizing Datapath Implementation Architectures with Parameters

Since AES-E128 performs 10 rounds on a plaintext block of 128 bits to complete the encryption, the algorithm inherently processes data in a batch of 128 bits. For the datapath architectures with the width of 8, 16, 32 and 64 bits, the round of AES-E128 is partially implemented in hardware and the reuse of the hardware for 16, 8, 4, and 2 times is required to complete one round, respectively. These architectures are called partial datapath architectures throughout this dissertation. When referring to a partial datapath architecture, the notataion Wn is used where n is the width of the datapath architecture. For the 128-bit width architectures, since the complete round function is implemented, they are called complete datapath architectures in this dissertation. Depending on the number of rounds implemented, a unrolling factor of 1, 2, 5 or 10 is used to characterize the complete datapath architectures, while the notation of Un is used to refer to a complete datapath architecture with the unrolling factor of n. When pipelined S-boxes are used in the datapath architectures, the notation of Gn or Pn is used to indicate the S-boxes are pipelined into n stages with gate level pipelining or component level pipelining, respectively (refer to Chapter 5 for details about gate level pipelining and component level pipelining).

The architectures of previous work on AES implementation can usually be characterized by the datapath parameters mentioned above. For example, [23] corresponds to U10. [20] includes U10, U10P2 and U10P3. [25] is U01. [26], [27], [28] and [29] are W32. [30], [31] and [32] are examples of W08. In this dissertation, it is assumed that the AES datpath architectures with the same datapath width, unrolling factor and/or pipelining stage numbers have the similar performance and the performance of the datapath architectures built in this dissertation is used to represent the performance of the architectures with the same parameters, including those proposed in previous work.

It should be noticed that the key expansion component and the controller for the AES implementation are not included in the datapath architectures that are considered in this dissertation. The reason is that, for a specific datapath architecture, it would not significantly vary with different implementations of the datapath architecture while the key expansion component and the controller can be highly flexible and independent on that datapath architecture. Especially for the key expansion component, which can be either implemented in order to generate the keys on-the-fly or not implemented so that the keys are generated offline and stored for the use by the datapath implementation.

2.3 Summary

In this chapter, the AES algorithm and the state-of-the-art hardware implementations of AES are reviewed and analyzed. Through the review and analysis, the potential for the increase of the implementation efficiency or the performance trade-offs is shown by exploring the architecture configurations of the S-box and the datapath. The research presented in Chapters 4 to 7 is the deployment of the investigation of the potential. As the foundation of the research work in this dissertation, the methodology used for the hardware implementation and performance evaluation in Chapters 4 to 8 is presented in the next chapter.

Chapter 3

Methodology of Hardware Implementation and Performance Evaluation

Since the research in this dissertation is based on the performance characterization of the hardware implementation of various AES architectures, the methodology used for building the hardware implementation of the AES architectures and evaluating the performance of the implementations is presented in this chapter.

3.1 Hardware Implementation

The hardware implementation performed in this dissertation is based on ASIC technology. The general digital ASIC design flow [33] is shown in Figure 3.1, which mainly involves four stages: algorithm modeling, RTL coding, synthesis and layout.

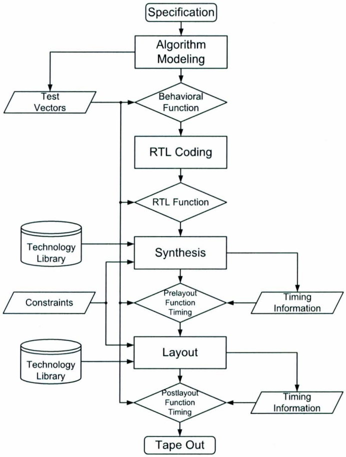


Figure 3.1: General digital ASIC design flow [53]

In this dissertation, the RTL description of the hardware implementation is coded with VHDL. The synthesis tool is Synopsys Design Compiler (DC, version X-2005.09) [34] with the 0.18-μm CMOS standard cell library from TSMC (for the work in Chapters 4 and 8) or DC (version B-2008.09) with the 90-nm CMOS standard cell library from STMicroelectronics (for the work in Chapters 5, 6 and 7). In this dissertation, the estimation of the performance of the implementations relies on the synthesis tool instead of the measurement of the fabricated circuit, so the physical layout stage in Figure 3.1 is skipped and, instead, an virtual layout process is performed in the topographical mode of DC during the synthesis process. The topographical mode of DC allows layout aware RTL synthesis, which can perform a coarse placement of the cells and extracts the interconnect resistance and capacitance from that and this process is called a virtual layout process [34]. With virtual layout, the parasitic capacitances of the physical layout of the design can be estimated more accurately compared with the wireload model-based statistical approximations. The estimation of capacitances ensures good correlation of the estimation in timing, area and power based on the synthesis result to that of the final physical design. For the virtual layout, the height-to-width ratio of the placement area and the area utilization [34] are set to 1 and 0.7, respectively, for all the implementations. The synthesis process generates the technology-dependent gate-level designs (netlists). The netlists are then used to estimate the timing, area, power consumption and energy consumption of the implementations. Although the estimation of the performance based on synthesis tool does not accurately reflects the realistic performance when the implementations are fabricated, all the implementations built in this dissertation are based on the same technology library and follow the same synthesis process, and therefore the relative comparisons should hold closely, leading to a fair and objective comparison.

3.2 Performance Evaluation

The performance of the implementations in this dissertation is evaluated in terms of timing, area, power consumption and energy consumption. In the next section, the evaluation of timing and area is presented firstly. After that, since the power consumption and the energy consumption of an implementation are distinguished in this dissertation, the basic concepts of power consumption and energy consumption is introduced before the consideration of their evaluation.

3.2.1 Evaluation of Timing and Area

In this dissertation, the timing of an implementation denotes the critical path delay. The critical path delay and area of an implementation are acquired from the synthesis report and the area is converted to Gate Equivalency (GE), where 1 GE is equal to the area of a two input NAND gate with the lowest drive strength in the technology library. Although the metric GE does not exactly reflect the size of the implementation after fabrication, the relative comparisons between different implementations should hold closely and these are of most interest in this dissertation.

3.2.2 Power Consumption and Energy Consumption

The total power consumption of a CMOS circuit consists of dynamic power consumption and static power consumption. Dynamic power consumption is the power consumed during gate switching, i.e., the output of the gate is changing. Static

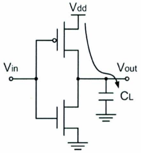


Figure 3.2: Charging current of the load capacitance of a CMOS inverter [55]

power consumption is the power consumed when the gate has voltage applied but is not switching.

3.2.2.1 Dynamic Power Consumption

The primary source of dynamic power consumption is switching power consumption, which is principally the result of charging of the load capacitance (output capacitance) of a gate [35]. The charging of the load capacitance of a CMOS inverter is illustrated in Figure 3.2. For each time the output of a gate changes from 0 to 1, the load capacitance is charged and the energy consumed is

$$Energy_{0\rightarrow 1} = C_L V_{dd}^2 \tag{3.1}$$

where C_L is the load capacitance and V_{dd} is the supply voltage. Then, the switching

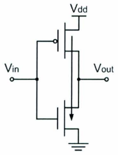


Figure 3.3: Short circuit current between supply power and ground of a CMOS inverter [55]

power can be described as

$$P_{switch} = C_L V_{dd}^2 Pr_{0\rightarrow 1} f_{clk} \tag{3.2}$$

where $Pr_{0\rightarrow 1}$ is the probability of the output switching from 0 to 1 and f_{clk} is the clock frequency.

In addition to switching power, internal power also contributes to dynamic power. Internal power is caused by the short circuit between power supply and ground when the PMOS and NMOS transistors are conducting simultaneously during the switching of the input, as shown in Figure 3.3. The energy consumed by the short circuit per switching period is

$$Energy_{short} = t_{sc} V_{dd} I_{peak} \tag{3.3}$$

where t_{sc} is the time both transistors are on and I_{peak} represents the short circuit current. Therefore, the internal power consumption can be described as

$$P_{\text{int}} = t_{sc} V_{dd} I_{peak} Pr_{switch} f_{clk} \tag{3.4}$$

where Pr_{switch} is the probability of the change in the input.

As a sum, the dynamic power can be expressed as

$$P_{dyn} = (C_L V_{dd}^2 Pr_{0\to1} f_{clk}) + (t_{sc} V_{dd} I_{peak} Pr_{switch} f_{clk}). \tag{3.5}$$

Since the internal power only occurs during the ramp time of the input signal of the gate which is normally very short, the overall dynamic power consumption is dominated by switching power [35], and therefore, the dynamic power consumption can be approximated to

$$P_{dyn} = C_L V_{dd}^2 Pr_{0\to1} f_{clk}. \tag{3.6}$$

According to the above expression, it is easy to see that, in the case of implementations with standard cell CMOS technology, $Pr_{0\to1}$ and f_{clk} are the factors that can be considered in the design and implementation to scale down the power consumption, while for full custom CMOS technology, more factors, including C_L and V_{dd} , can be taken into consideration for low power design.

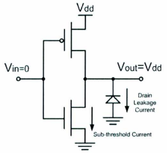


Figure 3.4: Static leakage currents of a CMOS inverter [55]

3.2.2.2 Static Power Consumption

Static power consumption in a CMOS gate can be expressed as

$$P_{stat} = I_{stat} V_{dd} \tag{3.7}$$

where I_{stat} is the total leakage current that flows between power supply and ground for the period the gate is not switching. Current I_{stat} consists of source or drain leakage current and sub-threshold current. Source or drain leakage current is the leakage current between the source or the drain and the substrate of a transistor, and sub-threshold current is the leakage current flowing from the drain to the source of a transistor operating in the weak inversion mode [35]. An illustration of these phenomena is shown in Figure 3.4.

Since all gates in a circuit suffer from static power consumption during non-switching period, it is obvious that static power consumption is proportional to the gate count of the circuit, i.e. the area of the circuit.

3.2.2.3 Energy Consumption

Energy consumption is the accumulated effect of power consumption over time, as expressed as

$$E = \int p(t)dt \tag{3.8}$$

where $p(t)$ is the power consumption at time t. The relation between energy consumption and power consumption can be illustrated by Figure 3.5 where the curves are instantaneous power consumption and the areas under the curves are energy consumption. The two approaches shown in Figure 3.5 have different power consumption ($P_1 > P_2$) over time but their total energy consumptions are same ($E_1 = E_2$). Under the assumption that a computation task of a device can be completed with either of the two approaches (i.e., in time T_1 for approach 1 and time T_2 for approach 2), approach 1 is preferred if the device is powered by a battery because the task is completed earlier ($T_1 < T_2$), and approach 2 should be selected if the device is passively powered and P_1 can not be well supplied. Consider now that using approach 1, it is possible to complete the task in a time $T_0 < T_1$. In this case, for energy constrained (i.e., battery powered) devices, it is clearly preferable to use approach 1, since less energy is used to complete the task, even though the instantaneous power consumption is higher than for approach 2.

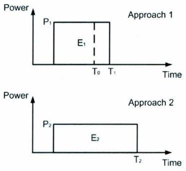


Figure 3.5: Power consumption and energy consumption

3.2.3 Evaluation of Power and Energy

The power consumption of the candidate implementations is estimated using PrimeTime PX (version B-2008.06-SP2) from Synopsys [36]. The estimation is a gate-level power analysis based on the the switching activity of the netlist. According to the definitions provided in the previous section, the dynamic power of a circuit can be calculated based on its power parameters (e.g., parasitic capacitances), timing information (e.g., clock frequency) and switching activity. The parasitic capacitances of the circuit are predicted and extracted in the synthesis and virtual layout process using the topology mode of DC. The switching activity is obtained from the gate-level simulation of the netlist with certain simulation vectors.

The power evaluation flow using PrimePower is shown in Figure 3.6. As is shown, after the synthesis stage, a forward Switching Activity Interchange Format (SAIF) [37] file is generated by Design Compiler as input to the simulation stage. The forward SAIF file contains the annotations about which circuit elements are to be traced during simulation. Then, the simulator runs gate-level simulation on the netlist with the simulation vectors that represent the typical tasks of the design. The switching activity during the simulation is captured and recorded in a backward SAIF file in the form of timing and toggle attributes of pins and ports. PrimePower calculates the power consumption using the backward SAIF file, the netlist file, the timing information, the parasitic information and the technology library. Since the calculation of power consumption by PrimePower heavily relies on the switching activity driven by the simulation vectors, a large number of random simulation vectors need to be generated to imitate the practical usage of the design in order to achieve accurate power estimation. PrimePower can report both the real-time power consumption of the circuit at any time point during the simulation period and the average power consumption over the period. With the report of real-time power consumption, the peak power consumption of the circuit is also known. Energy consumption is calculated by multiplying the average power consumption with the duration of the period of interest. In this dissertation, the energy is determined over a period of time for the implementation to produce one unit of throughput, such as a byte for the S-box implementation and a 128-bit block for the datapath implementation.

3.3 Summary

In this chapter, the methodology for the hardware implementation and the evaluation of the performance of the implementations in this dissertation is described. In summary, the ASIC design flow based on standard cell technologies is adopted for the hardware implementation. The performance of the hardware implementations is eval-

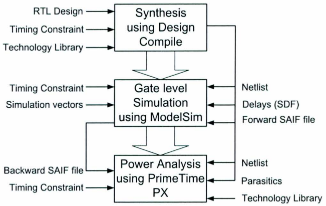


Figure 3.6: Power Evaluation flow using PrimePower

uated based on the netlist from the synthesis results of the RTL design for timing, area, power and energy. The estimated parasitic capacitances of the implementations from the virtual layout process are used for estimation of power and energy consumption.

Chapter 4

Using Pipelined AES S-Boxes for Resource Efficient Purpose: An Example

In this chapter, we propose a compact ASIC implementation for AES encryption with 128-bit keys which employs a single 4-stage pipelined S-box shared by the data path operation and the key expansion operation. Compared with the previous smallest encryption-only ASIC implementation of AES [31], it achieves an increase in throughput of 2.1 times while slightly reducing the gate count. This result indicates that pipelined AES S-boxes are applicable in AES hardware implementations targeted at low resource applications. The content of this chapter is also presented in [38].

4.1 Introduction

An AES encryption core with an 8-bit data path was presented in [31] where two S-boxes are implemented, one used by round operations and the other used by the key expansion. Even though the throughput of this design is higher than other compact designs, the critical path, which determines the maximum clock frequency and consequently the throughput, is quite long because it comprises the entire critical path of the S-boxes. S-boxes are the most complex component in an AES implementation and it generally involves a large number of gates on its critical path. Commonly in an AES implementation for high speed applications, the S-boxes are pipelined to several stages in order to reduce the critical path of the overall design. However this method is seldom applied to compact implementations for throughput improvement because it is assumed that pipeline registers would incur large hardware overhead, which is not affordable for the compact implementations targeted at low cost applications. In this chapter, the applicability of using a pipelined S-box in compact AES hardware implementations is examined. A new VLSI architecture design for AES implementation is proposed to accommodate a 4-stage pipelined S-box and the implementation results show that the new design can achieve more than double the throughput of [31] while slightly reducing the gate count. In the following of this dissertation, the design from [31] is referred to as the reference design.

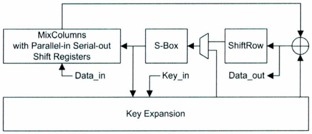


Figure 4.1: Block diagram of the proposed AES encryption core architecture

4.2 Architecture Design

The block diagram of the proposed architecture design is shown in Figure 4.1. In the architecture, the round operations have an 8-bit data path, and on the path, the *ShiftRows*, *SubBytes*, *MixColumns* and *AddRoundKey* operations are performed byte by byte in sequence by the corresponding components. To complete the operation of one round of AES encryption, all the bytes of the *State* need to traverse the round operation data path once, so totally 10 traversals are required to encrypt an 128-bit plaintext after the data path loads it. The key expansion component also has an 8-bit data path and generates round keys on-the-fly using 128-bit keys. One S-box is used alternately by round operations and the key expansion. During the period the S-box is occupied by the key expansion component, the round operations are frozen by clock gating. The proposed design is developed based on the reference design [31] and adopts the same *ShiftRows*, *MixColumns* and S-box structures. The proposed design has modified interconnection between components, key expansion component and data flow which allow the interleaved use of one pipelined S-box between the

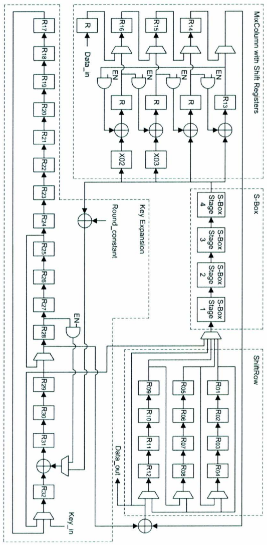


Figure 4.2: Architecture of the AES encryption core with a 4-stage pipelined S-box

data path operation and the key expansion operation. The detailed architecture of the proposed design is shown in Figure 4.2. All the paths in Figure 4.2 have a width of 8 bits except those between two consecutive pipeline stages in the S-box have the widths as the internal data widths of the S-box at the places where the pipeline registers are inserted. The blocks marked with "R" are 8-bit registers. The operation of each component and their interaction will be described in the following sections.

4.2.1 The *ShiftRows* Component

The *ShiftRows* component consists of 128-bit registers connected in series and there are shortcuts from its input and every fourth register to its output. The component takes bytes arriving in the order of *State* columns and reorders the bytes while they are passing through. The detailed operation of the component is described in [31].

4.2.2 S-Box

The S-box adopted in the proposed design is developed in [7]. Since the computation of multiplicative inverse over $GF(2^8)$ can be converted to the computations in subfields, in [7] the S-box structure is examined for a number of representations of subfields, including both polynomial bases and normal bases, and the one leading to the implementation with the smallest gate count is identified. In the proposed architecture, the S-box is pipelined to 4 stages. The pipeline registers are placed between two consecutive stages but not shown in Figure 4.2. The register placement is performed at the gate level. The exact register placement is not presented. Refer to Chapter 5 for the details.

4.2.3 The *MixColumns* Component

The *MixColumns* component is a serial-in, parallel-out matrix multiplier. It takes one byte input per clock cycle continuously for 4 cycles to receive a column of the *State*. At every fourth clock cycle, the computation of the *MixColumns* operation on the current column of the *State* is completed and the first byte of the result is available at the output while the remaining three bytes are fed to the input of the parallel-in, serial-out shift registers incorporated in the *MixColumns* component. Subsequently, the three bytes are shifted out in the following three cycles. The blocks $X02$ and $X03$ in Figure 4.2 generate the products of the current input byte and 02H and 03H, accordingly. The AND gates are used to bypass the XOR gates. This is done by setting EN to 0 and thus ensuring that the XOR operation does not change the data. During the loading of a 128-bit plaintext, only the shift registers at the right side of the component are working to shift in and shift out the plaintext bytes in serial. Refer to [31] for a detailed explanation of the component.

4.2.4 Key Expansion Component

The key expansion component has an 8-bit data path, which is implemented mainly by circularly connected shift registers R17 to R32. The bytes of a round key are generated while the key state circulates through the shift registers and the generation of a round key is completed every time all of the key state has circulated along the path once. The computation of the next round key involves the substitution of the last four bytes of the current round key. This is realized by an 8-bit multiplexer switching the input of the S-box between the round operation data path and the key expansion data path. During the load period of key bits, the AND gate has EN set to 0 to bypass the XOR gate on the shift register path.

4.2.5 Overall Design

In order to clarify the operation of the architecture, the states of the numbered registers in Figure 4.2 in certain selected clock cycles are shown in Table 4.1 and Table 4.2 for the round operation component and the key expansion component, respectively. For both tables, the output of the register during a clock period is regarded as the state of the register. In Table 4.1, for each state N_m $(0 \leq N \leq 10, 1 \leq m \leq 16)$, N represents the N-th round within which the byte of the *State* is processed (with the exception that $N = 0$ indicates the initial plaintext) and m represents the m-th byte of the *State* in the order of columns. The state N_m represents the m-th byte of the *State* after the $MixColumns$ operation in the N-th round (after the *SubBytes* operation in the final round). Similarly, in Table 4.2 the state of a register N_m indicates the m-th byte of the N-th round key with the original key bits represented with $N = 0$. In both tables, X indicates a state holding a useless byte. The operation of the multiplexers and the AND gates can be easily determined from Table 4.1 and Table 4.2. Clock gating is applied regularly to both round operation and key expansion components. The selected cycles that demonstrate the occurance of clock gating are marked with * in Table 4.1 and Table 4.2. The registers that require clock gating and the cycles when clock gating is active can be deduced from Table 4.1 and Table 4.2. It should be mentioned that, as is shown in Table 4.1 and Table 4.2, the architecture works in a way for the final round operations slightly differently from that for other rounds because the *MixColumns* operation is skipped in the final round. It takes 256 clock cycles to complete the encryption of a 128-bit plaintext including loading and unloading, and since there is overlapping of three clock cycles during loading and unloading, the effective clock count of the architecture is 253 for the encryption of a 128-bit block of plaintext.

4.3 Implementation Results and Discussion

The proposed AES architecture design with a 4-stage pipelined S-box is synthesized using Synopsys Design Compiler version X-2005.09 under 0.18-μm CMOS standard cell technology from TSMC. The synthesis results of the proposed design with the constraint of minimum area are reported in Table 4.3. Since it is difficult to compare performance of implementations with different technologies, the implementation results under 0.13-μm technology from [31] are not quoted for comparison here, and instead, the reference design is implemented and synthesized with the same tool and technology as the proposed design. The results are presented in Table 4.3. It can be seen that the design with the pipelined S-box uses slightly fewer gates than the reference design and achieves an increase in throughput by a factor of 2.1. Although the overhead of control logic is not included in the comparison, the slight increase in gates used for the controller of the proposed design would be countered by the slight decrease in gates on the data path. The implementation results and comparison show that, even though the pipelined S-box would introduce the latency of several clock cycles per round operation compared with the reference design, the reduction of the critical path delay by using the pipelined S-box compensates for the increased latency

Table 4.1: Register states of the round operation data path

Cycle	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}	R_{15}	R_{16}
2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0_1
15	X	1_1	1_2	1_3	1_4	1_5	1_6	1_7	1_8	1_9	1_{10}	1_{11}	X	1_{12}	1_{13}	1_{14}
*16-20	1_1	1_2	1_3	1_4	1_5	1_6	1_7	1_8	1_9	1_{10}	1_{11}	1_{12}	X	1_{13}	1_{14}	1_{15}
21	1_2	1_3	1_4	1_5	1_6	1_7	1_8	1_9	1_{10}	1_{11}	1_{12}	1_{13}	X	1_{14}	1_{15}	1_{16}
28	1_9	1_2	1_3	1_{12}	1_{13}	1_{14}	1_7	1_8	X	X	X	X	$1'_1$	$1'_2$	$1'_3$	$1'_4$
29	1_2	1_3	1_{12}	1_{13}	1_{14}	1_7	1_8	X	X	X	X	2_1	$1'_2$	$1'_3$	$1'_4$	$1'_5$
*41-44	2_1	2_2	2_3	2_4	2_5	2_6	2_7	2_8	2_9	2_{10}	2_{11}	2_{12}	$1'_{13}$	$1'_{14}$	$1'_{15}$	$1'_{16}$
*232-236	10_1	10_2	10_3	10_4	10_5	10_6	10_7	10_8	10_9	10_{10}	10_{11}	10_{12}	$9'_{13}$	$9'_{14}$	$9'_{15}$	$9'_{16}$
241	10_2	10_7	10_8	10_9	10_{10}	10_3	10_{12}	10_{13}	10_{14}	10_{15}	10_4	X	$10'_1$	X	X	X
256	X	X	X	X	X	X	X	X	X	X	X	X	$10'_{16}$	0_1	0_2	0_3

Table 4.2: Register states of the key expansion component

Cycle	R_{17}	R_{18}**......**R_{24}	R_{25}	R_{26}	R_{27}	R_{28}	R_{29}	R_{30}	R_{31}	R_{32}
1	X	X	X	X	X	X	X	X	X	X
17	0_{2}	0_{3} ...... 0_{9}	0_{10}	10_{11}	10_{12}	10_{13}	10_{14}	10_{15}	10_{16}	0_{1}
20	0_{5}	0_{6} ...... 0_{12}	0_{13}	10_{14}	10_{15}	10_{16}	0_{1}	0_{2}	0_{3}	0_{4}
*21	0_{5}	0_{6} ...... 0_{12}	0_{14}	0_{15}	0_{16}	0_{13}	0_{2}	0_{3}	1_{4}	0_{1}
*23	0_{5}	0_{6} ...... 0_{12}	0_{16}	0_{13}	0_{14}	0_{15}	1_{4}	1_{1}	1_{2}	0_{3}
*24-28	0_{5}	0_{6} ...... 0_{12}	0_{13}	0_{14}	0_{15}	0_{16}	1_{1}	1_{2}	1_{3}	1_{4}
29	0_{6}	0_{7} ...... 0_{13}	0_{14}	0_{15}	0_{16}	1_{1}	1_{2}	1_{3}	1_{4}	0_{5}
31	0_{8}	0_{9} ...... 0_{15}	0_{16}	1_{1}	1_{2}	1_{3}	1_{4}	1_{5}	1_{6}	0_{7}
44	1_{5}	1_{6} ...... 1_{12}	1_{13}	1_{14}	1_{15}	1_{16}	1_{1}	1_{2}	1_{3}	0_{4}
*45	1_{5}	1_{6} ...... 1_{12}	1_{13}	1_{14}	1_{15}	1_{16}	1_{2}	1_{3}	2_{4}	1_{1}
*237	9_{5}	9_{6} ...... 9_{12}	9_{13}	9_{14}	9_{15}	9_{16}	9_{2}	9_{3}	10_{4}	9_{1}
*240	9_{5}	9_{6} ...... 9_{12}	9_{13}	9_{14}	9_{15}	9_{16}	10_{1}	10_{2}	10_{3}	10_{4}
*241	9_{5}	9_{6} ...... 9_{12}	9_{13}	9_{14}	9_{15}	9_{16}	10_{1}	10_{2}	10_{3}	10_{4}
256	10_{7}	10_{8} ...... 10_{11}	10_{12}	10_{13}	10_{14}	10_{15}	10_{16}	10_{1}	10_{2}	10_{3}

Table 4.3: Implementation results

	Area (GEs)	Max. Freq. (MHz)	Clock Cycles per block	Max. Throughput (Mbps)
Proposed	2730	233	253	117.9
Reference design [31]	2815	69	160	55.2

Table 4.4: Normalized performance comparison of the architecture using a single S-box with different number of stages

# Pipeline Stages	1	2	3	4	5
Area	0.93	0.96	0.99	1	1.05
Throughput	0.37	0.65	0.82	1	1.15
Ratio (Throughput/Area)	0.40	0.65	0.83	1	1.1

and brings significant boost to the throughput. Therefore, when throughput is a concern for a low gate count AES hardware implementation, the proposed design with a pipelined S-box is a much better choice than the reference design with two S-boxes. Only the performance comparison with the reference design is presented here because the reference design uses the lowest hardware cost among published works based on an ASIC platform [31].

In order to determine the influence of the number of pipeline stages on the overall performance of a compact design, the scenarios for varying number of pipeline stages are investigated. The area and throughput performance of the architecture using a single S-box with a variety of pipeline stages is normalized to the 4-stage pipeline scenario and shown in Table 4.4. It should be noted that the architecture in Figure 4.1 only works with a 4-stage pipelined S-box and for other stage numbers up to 5 the architecture requires minor changes to fit. For more than 5 stages of pipeline, a major modification on the architecture is required since it is getting complicated to share the S-box between the datapath and the key expansion when the latency of the S-box becomes larger than 5 clock cycles. The differences in area between pipeline stage numbers come from the different amount of pipeline registers used in each case. The figures of one pipeline stage in Table 4.4 indicate the scenario of using a non-pipelined S-box. From Table 4.4, it can be seen that the ratio of throughput/area is gradually improved as the number of the pipeline stages increases. The architecture with a 4-stage pipelined S-box is selected to be specified in this chapter because it has the best performance for an architecture with an area smaller than the reference design [7]. The information in Table 4.4 also indicates the performance variance over different numbers of the pipeline stages of the S-box implementation is worth investigation in order to identify the appropriate number of pipeline stages for certain design requirements. In the next chapter, a detailed and comprehensive investigation on the pipeline configurations of the AES S-box is performed through the characterization of the performance of the different pipelined S-box implementations under a variety of throughput requirements with a 90-nm CMOS standard cell technology.

4.4 Summary

In this chapter, a new architecture design for compact hardware implementation of an AES encryption core is presented. The new design is featured with a 4-stage pipelined S-box. The implementation results show that, compared with the previous smallest encryption-only AES hardware implementation, the new design uses the same amount of gates to achieve an increase of 2.1 times in throughput. The implementation results indicate that not only are pipelined S-boxes are applicable to compact implementations of AES, they can actually be used to improve performance. The content of this chapter is a preliminary attempt to improve the performance of the AES implementation with pipelined S-boxes. Table 4.4 also shows some preliminary results of the exploration of the performance trade-offs with the different number of pipeline stages. Inspired by these results, in the next chapter, a comprehensive investigation of the performance improvement and trade-offs provided by pipelined AES S-boxes is performed and presented.

Chapter 5

Exploration of S-Box Pipeline Configurations for Flexible and Efficient Implementation

In this chapter, we present a comprehensive investigation of the pipeline configurations for ASIC implementations of the Advanced Encryption Standard (AES) substitution box (S-box). We consider pipeline configurations for the S-box with a typical composite field structure by varying the number of pipeline stages and the placement approach of pipeline registers. Besides the conventional placement approach at the component level of the S-box, we adopt a new placement approach at the gate level to achieve a fine-grained pipeline. The characterization shows that there is notable performance improvement in timing, area, power and/or energy efficiency by using an appropriate configuration compared with other configurations including non-pipelined implementations.

5.1 Introduction

In contrast to the conventional usage of pipelined S-boxes in high throughput implementations, there are very few previous works that investigate the potential of pipelined S-boxes for resource efficiency so that they can be applied to resource-constrained applications, such as lightweight embedded applications. This is due to the fact that the area overhead introduced by the pipeline registers appears to conflict with the effort of reducing area and consequently reducing power and energy consumption.

In the next section, based on a typical composite field structure of the AES S-box, we consider an extensive variety of pipeline configurations and investigate their influence on all perspectives of the performance, including timing, area, power and energy. The pipeline configurations consist of the number of pipeline stages of 2 to 4 for the component level register placement approach and 2 to 7 for the gate level register placement approach. We introduce the application of the gate level approach for the pipeline of S-box implementations. It exploits the retiming function by the synthesis tool to achieve the fine-grained pipeline at the gate level without the violation of the standard ASIC design flow. The performance of the pipelined S-box implementations is benchmarked with a 90-nm CMOS standard cell technology.

Through the analysis of the performance, some obvious trends that reflect the influence of the pipeline configurations on the performance are identified. In addition to the timing improvement, notable improvements in terms of area, power and energy efficiency are also observed by using an appropriate pipeline configuration compared with implementations with no pipeline. These improvements occur under the wide range of timing requirements we have examined, including the requirements to which lightweight AES implementations are targeted. These results are strong evidence that pipelined S-box implementations are not only suitable for high throughput AES implementations, but also valuable to resource-efficient AES implementations. The results also show that pipelining provides many more performance options that allow more flexible implementation of the AES S-box compared with non-pipelined implementations.

5.2 The S-box Structure for Pipelining

As is mentioned in Section 2.2.1, the composite field structures have low complexity and this allows for more efficient pipelining in terms of the number of pipeline registers. Therefore, the composite field structures are most suitable for pipelining.

Although quite a number of composite field S-box structures are available, including [5], [6], [7], [8], [9], [10] and [13], there is typically little difference between them. They tend to be very close together in performance compared with the other groups of S-box structures [18]. Since our purpose is to investigate the general effectiveness of pipelining the S-box with a composite field structure, we use the one from [5] as a typical and common composite field structure for the study in this chapter. Although it is not the one with the best performance among all the composite field structures, it has a relatively simple and clear structure and can be easily pipelined. The pipeline of the S-box in [19] and [20] is also based on this structure. Further, since the S-boxes for encryption and for decryption have very similar structure, we focus our investigation on the encryption S-box.

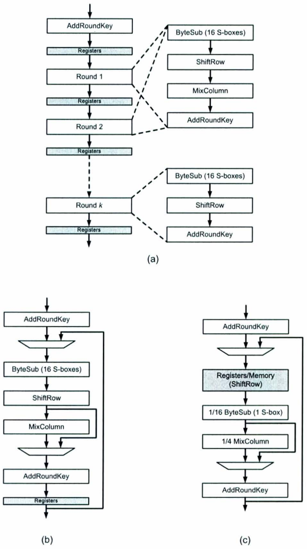


Figure 5.1: The typical data path architectures of the AES: (a) loop-unrolled architecture, (b) round-iterative architecture, (c) fully serialized architecture

5.3 Applicability of the Pipelined AES S-Box

When considering replacing non-pipelined S-boxes with pipelined S-boxes in an AES implementation, the primary concern is whether it would impact the throughput of the cipher. Actually, the answer to this concern varies with the contexts, including the architecture of the cipher and the mode of operation of the cipher. In the next section, the impact on the throughput will be discussed in the contexts of three typical data path architectures of AES and the two groups of the commonly used block cipher modes of operation: the non-feedback modes and the feedback modes.

Three typical AES hardware architectures are the loop-unrolled architecture [19] [20], the round-iterative architecture [39] and the fully serialized architecture (with a datapath width of 8 bits) [31] [32], and the illustration of them is shown in Figure 5.1. The loop-unrolled architecture is usually adopted for high throughput implementations while the fully serialized architecture targets at low area implementations. The round-iterative architecture provides a trade-off option in between. Non-feedback modes include electronic codebook (ECB) mode and counter (CTR) mode, and feedback modes include cipher-block chaining (CBC) mode, cipher feedback (CFB) mode and output feedback (OFB) mode. In each context, two scenarios are considered after the replacement of the non-pipelined S-boxes with the pipelined S-boxes: (a) the cipher runs at the same clock frequency, and (b) the cipher runs at higher clock frequency. For convenience, the assumption is made that the inputs and the round keys are able to be fed to the data path whenever they are required.

The loop-unrolled architecture is developed particularly for the non-feedback modes in order to fully exploit its capability to produce a 128-bit output per clock cycle and, therefore, it is assumed that this architecture is only used for the non-feedback modes. For scenario (a), there is no impact on the throughput after the replacement. For scenario (b), the throughput would increase.

For the round-iterative architecture working in the non-feedback modes, the throughput can be kept the same (scenario (a)) or increased (scenario (b)), due to the fact that the number of inputs that can be processed simultaneously is equal to the number of pipeline stages. In the cases of the feedback modes, there is only one input that can be processed at one time due to the dependency between the current input and previous output, and consequently the throughput drops by a factor of the number of pipeline stages for scenario (a). However, if this cipher works under one of the feedback modes but serves for parallel independent data streams/channels with the number equal to or larger than the number of pipeline stages, as is also assumed in [40], the throughput can be kept the same for scenario (a) or increased for scenario (b).

The fully serialized architecture with one S-box takes at least 16 clock cycles to complete one round function and iterates for the number of rounds to produce one input. It is not able to handle more than one input at a time, so there is no difference caused by the mode of operation in evaluating the impact on throughput by pipelining. Assuming that the fully serialized architecture takes 16 clock cycles to complete one round function and leads to the throughput T before an n-stage pipelined S-box ($n \geq 2$) is used, the number of clock cycles for one round function becomes $16+(n-1)$ after the replacement and the throughput becomes $16T/(15+n)$ in scenario (a). Considering that the fully serialized architecture is usually used in the applications running at low clock frequency, the decline of the throughput after the replacement is slight and acceptable if the pipelined S-box has a small number of stages. For scenario (b), the throughput can be increased above $16T/(15+n)$ by increasing the clock frequency. It will be shown in the following that there is no or very little penalty on the area and energy efficiency if the clock frequency is increased for the S-box implementation built for running at low clock frequency.

In this work, the pipelined S-boxes implementations are compared mostly under scenario (a), so that the throughput of the S-box remains the same for pipelined and non-pipelined implementations, regardless of the number of pipeline stages. Based on the above analysis, this context is prevalent.

5.4 Pipelining the AES S-Box

S-box implementations with the composite field structures create a long critical path delay for the overall AES implementation. For most AES implementations, the critical path of the overall implementation lies in the hardware of one round function. A round function would perform the operations *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey* in sequence in the data path part. Among these operations, *MixColumns* can be built with 3 XOR gates on its critical path [19], *AddRoundKey* is single XOR gates in parallel, and *ShiftRows* is a crossover of wiring in hardware. Therefore, the critical path of the S-box takes up almost the whole critical path of the overall implementation. Pipelining the S-box would effectively reduce the critical path delay of the overall implementation. In this section, we investigate the possible pipeline configurations applicable to the S-box with a composite field structure. We refer to a pipeline configuration as a combination of a placement approach of pipeline registers and a number of pipeline stages.

Another benefit of pipelining an S-box with long critical path delay is the reduction of glitches in the circuit and consequently the reduction of power and energy consumption. Glitches are the unnecessary transitions of signals in a circuit. They are generated when the input signals do not arrive simultaneously at a gate and these glitches may propagate to generate more glitches throughout the circuit. Since the dynamic power consumption of a circuit is proportional to the number of transitions in the circuit and dominates the total power consumption of a circuit in the CMOS process, the negative influence on the power consumption caused by glitches can be severe if the amount of glitches is large. According to the transition simulation performed in [41] on a direction detector that has a long critical path delay, about 80% of the total transitions in the circuit are glitches. An effective approach to reduce glitches is to insert flip-flops in the circuit, as is shown in [42]. The work in [41] also shows that there is significant reduction in power consumption if the optimal number of flip-flops is used. Since the AES S-box with a composite field structure is also a combinational circuit with long critical path delay, it is reasonable to believe that pipelining can contribute to the power and energy efficiency of the S-box implementation through the reduction of glitches and this is confirmed by the characterization results in this chapter.

5.4.1 Pipelining at the Component Level

Pipelining at the component level is a coarse-grained placement approach. An example that pipelines the S-box structure of [5] into 2, 3 and 4 stages is shown in Figure

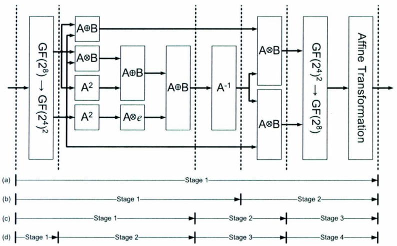


Figure 5.2: Component level pipelined S-box architectures: (a) 1-stage (no pipeline), (b) 2-stage, (c) 3-stage, (d) 4-stage

5.2, where A and B are elements in $GF(2^4)$, $\oplus$ and $\otimes$ are addition and multiplication over $GF(2^4)$, respectively, and e is the constant in hexadecimal notation. The dotted lines represent the positions of the registers for the corresponding number of pipeline stages indicated at the bottom. The 2-stage and 3-stage pipelining are the same as used in [20]. Pipelining at the component level can be realized at the Register Transfer Level (RTL) in the HDL description of the S-box. The registers are placed according to the estimated complexity or delays of the components before the S-box is synthesized into the gate-level design. For this placement approach, the number of pipeline stages is examined up to 4 in this chapter since there becomes severely unbalanced critical path delays in the stages when the number is larger.

5.4.2 Pipelining at the Gate Level

A fine-grained placement approach for pipelining may be desired since it can avoid the problem of unbalanced delays. The most fine-grained placement can be achieved by pipelining at the gate level.

In the ASIC design flow using standard cell libraries, a gate level design, also know as a netlist, is generated from the synthesis process applied to the RTL design. It should be noted that inserting registers into the RTL design, as is done for the component level pipeline, would not lead to a desirable gate level pipeline with the well balanced delays since the delays can not be accurately estimated before the synthesis. On the other hand, after the synthesis, the manual insertion of registers into the netlist for gate level pipeline is not compliant with the standard ASIC design flow and could violate the optimization effort of the synthesis, considering that the manual pipeline would impose a major modification of the netlist that has been well optimized by the synthesis towards synthesis constraints. To get rid of this dilemma, we adopt the retiming function of the synthesis tool to perform the gate level pipeline during the synthesis process. In this way, a non-pipelined RTL design of the S-box would generate the gate level pipelined netlist after the synthesis process. The details of the implementation of this approach in our experiment are described in Section 5.5.1.

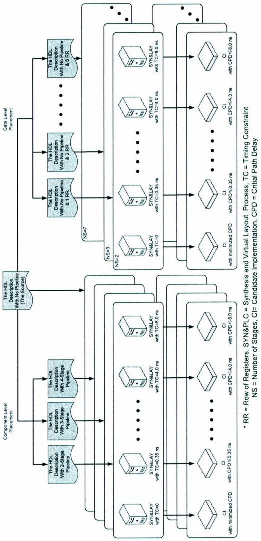
For the gate level approach, theoretically, the critical path delay can be shortened to the minimum by increasing the number of pipeline stages until there is only a single gate left on the critical path of a stage. When the number of stages becomes large and the gate delay on the critical path is close to the delay of the register, there is little or no improvement of timing from more pipeline stages. In this chapter, the number of pipeline stages is examined up to 7 for the gate level approach in order to observe the saturation in timing improvement.

5.4.3 Comparing Placement Approaches

Despite its superiority in terms of balanced delays, the gate level placement approach is inferior in terms of the number of pipeline register bits required. Compared with the component level approach that only places register bits on the input or output ports of a component, the gate level approach usually requires a expanded number of bits for pipelining when the components are decomposed into gates. On the other hand, the flexibility in register placement of the gate level approach allows the synthesis process to have more optimization potential. Therefore, there is no straightforward way to tell which approach leads to overall better performance on the implementations until a comprehensive characterization and comparison is conducted, as is done in this chapter.

5.5 Methodology

Although the number of pipeline stages is the essential factor determining the critical path delay of pipelined S-box implementations, the synthesis tool can tune the implementations with different pipeline configurations to meet the same timing constraint. In this way, they can be compared to reflect the performance variation while varying the pipeline configuration to identify the appropriate configurations that lead to the most desired performance for different timing constraints. The S-box implementa-



Component Level Placement
Gate Level Placement
The HDL Description With No Pipeline (The Source)
The HDL Description With 2-Stage Pipeline
The HDL Description With 3-Stage Pipeline
The HDL Description With 4-Stage Pipeline
The HDL Description With No Pipeline & 1 RR
The HDL Description With No Pipeline & 2 RR
The HDL Description With No Pipeline & 6 RR
NS=2
NS=3
NS=7
SYN&LAY with TC=0
SYN&LAY with TC=0.35 ns
SYN&LAY with TC=4.0 ns
SYN&LAY with TC=8.0 ns
CI with minimized CPD
CI with CPD≤0.35 ns
CI with CPD≤4.0 ns
CI with CPD≤8.0 ns
* RR = Row of Registers, SYN&PLC = Synthesis and Virtual Layout Process, TC = Timing Constraint
NS = Number of Stages, CI= Candidate Implementation, CPD = Critial Path Delay

Figure 5.3: Derivation of the candidate implementations from the source HDL description of the S-Box

tions of different pipeline configurations are referred as *candidate implementations* in the next sections. The candidate implementations are tuned and compared against the same timing constraints because timing usually has the first priority as a design requirement. This is the basic methodology we adopt for the investigation of pipeline configurations.

5.5.1 Deriving the Candidate Implementations

The derivation of all the S-box implementations based on the source HDL description of the non-pipelined AES S-box is shown in Figure 5.3. The source HDL description of the non-pipelined S-box in Figure 5.3 is the structural description of the S-box with the composite field structure from [5], and the structure is shown in Figure 5.2. Both the input and output ports in this HDL description are registered, as is indicated by the dotted line at the input and output of the S-box in Figure 5.2.

In Figure 5.3, three categories of the HDL descriptions are derived firstly from the source HDL description and then each of the HDL descriptions is used to generate a number of the implementations under different timing constraints. Each of the HDL descriptions reflects a pipeline configuration. Each of the implementations is generated from a synthesis and virtual layout process based on the HDL description.

The first category of HDL descriptions contains that of the non-pipelined S-box, which is exactly the same as the source HDL description. It is used to generate the non-pipelined candidate implementations as the references. The second category contains the HDL descriptions of the pipeline configurations with the component level placement approach. They are derived by inserting the registers as D-flip flops into

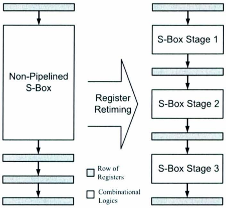


Figure 5.4: Illustration of the gate level approach of pipelining into 3 stages by register retiming

the source HDL description according to Figure 5.2. The third category is the pipeline configurations with the gate level placement approach.

The gate level approach is performed with the register retiming function provided by Synopsys Design Compiler [34]. The register retiming function moves registers through the combinational logic of a design to optimize timing and area. An illustration of how this function works is shown in Figure 5.4. Before the register retiming, rows of registers need to be inserted in the source HDL description of the S-box according to the target number of stages and they can be simply placed at the outputs of the S-box. Each row of registers initially has a width of 8 bits and while it is moved to the appropriate position during the register retiming, the width will be adjusted to fit the width of the data path at the position. It should be noted that although the design produced by the retiming process is subject to formal verification, the correctness of our pipelined S-boxes can be easily verified by a exhaustive test. Since the retiming process is automatic and the placement of registers may vary from one candidate implementation to another, the detailed locations of the pipeline registers are not presented in this chapter.

For each pipeline configuration, a number of different timing constraints are used in the synthesis process to produce different candidate implementations. These timing constraints consist of minimized delay (timing constraint is set to 0) and some selected constraints covering a wide range of timing requirements. These selected timing constraints include 0.35 ns, 0.40 ns, 0.60 ns, and 0.80 ns as the tight constraints, 1.00 ns, 1.25 ns, 1.50 ns, 1.75 ns and 2.00 ns as the medium constraints, and 2.50 ns, 3.00 ns, 4.00 ns and 8.00 ns as the loose constraints. A candidate implementation is omitted if it cannot meet its target timing constraint.

The synthesis and virtual layout process is performed in the topographical mode of Synopsys Design Compiler, as is described in Chapter 3. The synthesis library is the 90-nm CMOS standard cell library from STMicroelectronics with a core voltage of 1.2V and standard threshold voltage.

5.5.2 Evaluation of the Performance

After the candidate implementations are achieved from the above process, their performance in terms of critical path delay, area, power and energy is estimated according to the performance evaluation methodology described in Chapter 3. The power consumption is estimated at the clock frequency corresponding to its target timing constraint, and this may differ from the maximum clock frequency at which it is able to work. For the candidate implementations that are built under the constraint of minimized delay, the maximum clock frequency is used for the estimation. The energy consumption of a candidate implementation is calculated as the product of its power consumption and the clock period applied to it. This energy consumption is normalized to be the average energy consumed to produce one byte output.

The power consumption of the candidate implementations is estimated using PrimeTime PX from Synopsys. The switching activity is obtained from the gate-level simulation of the netlist with 10,000 random byte inputs. The same set of random byte inputs is used for all the candidate implementations.

5.6 Experimental Results and Analysis

Following the methodology described in the last section, there are 10 HDL descriptions derived from the source HDL description and totally 127 candidate implementations are built from them. We investigate the effect of the pipeline configurations upon the performance of the candidate implementations in three ways. Firstly, each pipeline configuration has its performance under different timing constraints examined. Secondly, the candidate implementations from different pipeline configurations with the minimized timing constraint are compared. Thirdly, the candidate implementations from different pipeline configurations with given timing constraints are compared.

5.6.1 Performance versus Timing Constraints

The normalized area, power and energy of the candidate implementations with the same pipeline configuration are placed in the same subfigure of Figure 5.5 according

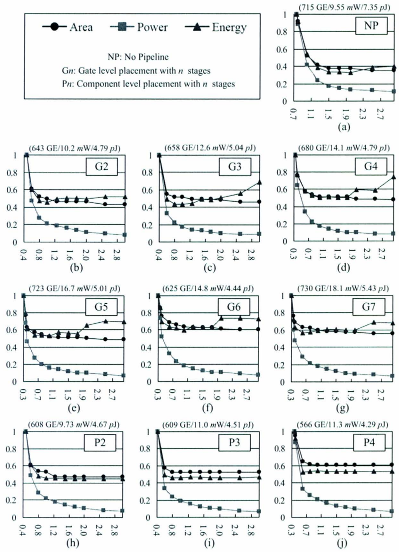


Figure 5.5: Normalized performance versus target timing constraints (ns), grouped according to the pipeline configuration

to their timing constraints. The pipeline configuration of the component level or the gate level placement approach with the pipeline of n stages is denoted as Pn or Gn, respectively, and the non-pipelined configuration is denoted as NP. The area, power and energy costs are normalized with respect to the leftmost values, which are shown at the top of the subfigure, respectively. The performance of the candidate implementations with the timing constraints of 4.00 ns and 8.00 ns is not shown in Figure 5.5 because they have the same area and negligible difference in power and energy compared with the implementations at 3.00 ns.

According to Figure 5.5, all the pipeline configurations have similar trends in area, power and energy with the variation of the timing constraint. The trends reflect the trade-offs between timing and the resource costs. When the timing constraint is loosened, the area of the pipeline configuration drops until the minimal is reached. In Figure 5.5, each of the pipeline configurations has the candidate implementation with the minimal area under a certain timing constraint and for the further loosened timing constraints, the area remains same. Another common feature shared by almost all the pipeline configurations is that the trend in energy does not exactly follow the area trend. While the area is being reduced to the minimum, there is a noticeable increase in energy. This indicates that the implementation with minimal area does not lead to the minimal energy consumption. For each pipeline configuration, the candidate implementation that has the minimum energy indicated in Figure 5.5 can be used at the lower clock frequencies for better energy efficiency than those candidate implementations achieved for the corresponding timing constraints. These candidate implementations lead to the energy-wise costs of the pipeline configurations that are examined in Section 5.6.3.

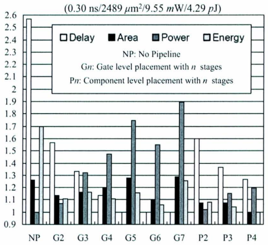


Figure 5.6: Normalized performance versus pipeline configurations under the synthesis constraint of minimized critical path delay

5.6.2 Performance versus Pipeline Configurations for the Minimized Timing Constraint

When the throughput of the S-box implementation is expected to be as high as possible, the implementations with the minimized timing constraint from different pipeline configurations are considered. Figure 5.6 shows the performance of these candidate implementations. The delay, area, power and energy in Figure 5.6 are normalized with respect to the corresponding minimum values, respectively, which are shown at the top of the figure.

As expected, the critical path delay decreases with the increase of the number of pipeline stages until it reaches the minimum for the pipeline configuration G5. After G5, the delay remains the same for G6 and G7 and this indicates that the saturation of timing improvement is reached, as is anticipated in Section 5.4. Among G5, G6 and G7 that have the same minimal delay, G6 is considerably better then the other two in area, power and energy. Therefore, increasing the number of pipeline stages beyond 6 does not improve the performance in timing, area, power or energy.

Directly comparing pipeline placement approaches, it can be seen that the gate level approach leads to shorter delays, while the component level approach leads to smaller area and less power and energy through comparisons between G2 and P2, G3 and P3, and G4 and P4. This confirms the conjecture in the analysis of the two approaches in Section 5.4 that the gate level approach has the advantage of the well balanced critical path delay, while the component level approach requires fewer pipeline register bits (due to the datapath width at the component ports) and this contributes to efficiency in area, power and energy.

5.6.3 Performance versus Pipeline Configurations for Given Timing Constraints

In the circumstance that a given throughput is expected from the S-box implementation, the performances of the pipeline configurations with the same timing constraint are compared, as is shown in Figures 5.7, 5.8 and 5.9. The timing constraint as well as corresponding clock frequency are presented at the top of each subfigure of Figures 5.7, 5.8 and 5.9. The area, power and energy costs are normalized with respect to the leftmost values, respectively, that are shown at the top of the subfigure. Since energy is the same metric as power under a given clock frequency, they share the same bars in Figures 5.7, 5.8 and 5.9.

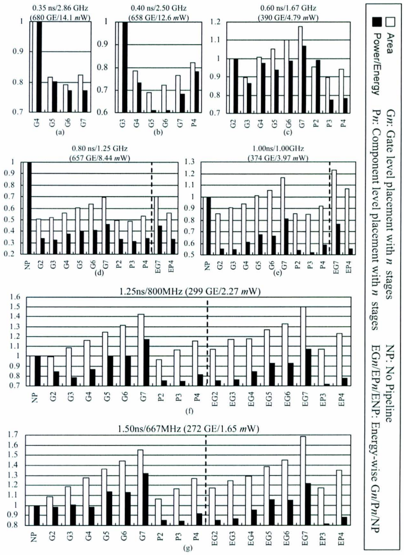


Figure 5.7: Normalized performance versus pipeline configurations under given timing requirements (from 0.35 ns/2.86 GHz to 1.50 ns/667 MHz)

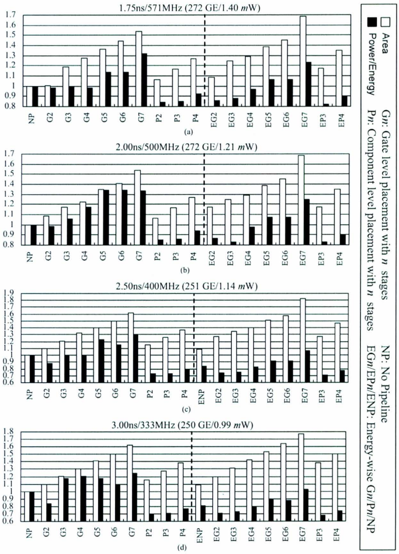


Figure 5.8: Normalized performance versus pipeline configurations under given timing requirements (from 1.75 ns/571 MHz to 3.0 ns/333 MHz)

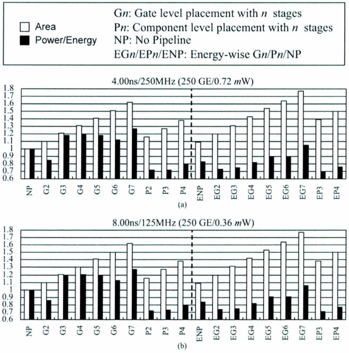


Figure 5.9: Normalized performance versus pipeline configurations under given timing requirements (4.0 ns/250 MHz and 8.0 ns/125 MHz)

According to the discussion in Section 5.6.1, for a given pipeline configuration, the most energy efficient implementation is generated with modest but not necessarily the loosest timing constraint. These candidate implementations are estimated for power/energy at the clock frequencies lower than the frequency corresponding to their timing constraints. Their power/energy consumption, if applicable, is reported at the right in the subfigures according to the actual frequency, along with the areas of these implementations, and are marked as the "Energy-wise" costs of the pipeline configurations. To distinguish from the energy-wise costs, the costs shown in the left part of the subfigures are called regular costs.

In the next section, we analyze the variance of the area and power/energy costs for different pipeline configurations under the various timing constraints based on Figures 5.7, 5.8 and 5.9.

5.6.3.1 Area versus the Number of Pipeline Stages

The area cost roughly follows the trend that, for a tight timing constraint, the increase of pipeline stages gradually reduces the cost until the minimal is reached by an appropriate number of stages. This trend is reflected in the cases from 0.35 ns/2.86 GHz to 1.25 ns/800 MHz. As the timing constraint is loosened to medium and below, the cost gets larger with the increase of pipeline stages, as is shown in the remaining cases of Figures 5.7, 5.8 and 5.9, where no pipeline leads to the minimal area cost.

This trend of area cost varying with timing constraints is reasonable because the increase of pipeline stages could relieve the timing constraint imposed in each stage and consequently improve the area efficiency for the tight timing constraint. Although the increase of stages would raise the area cost by introducing more pipeline registers, there could be overall gain and the appropriate number of pipeline stages maximizes the gain. When the timing constraint is further loosened, the effect on area reduction becomes less significant compared with the area increase by more pipeline stages and therefore more stages would no longer reduce the area cost. Finally, when it is realizable for the given timing constraint, the non-pipelined implementation results in the minimal area and the more pipeline stages, the more area is incurred.

5.6.3.2 Power/Energy versus the Numbers of Pipeline Stages

By looking at the regular power/energy values (excluding those energy-wise values) in Figures 5.7, 5.8 and 5.9, it is able to see the trend in power/energy efficiency is very similar to that of area efficiency and the reasoning follows similarly. The only obvious difference between the trends is that pipelined implementations leads to better power/energy efficient implementation than non-pipelined implementations even for the medium to low throughput cases, as is shown in the cases from 1.50 ns/667 MHz to 8.0 ns/125 MHz. This can be attributed to the reduction of glitches by using the pipeline registers. It is supported by our experiment that, when the pipeline registers of P2 in the cases of 4.0 ns/250 MHz and 8.0 ns/125 MHz are manually removed, the power and energy increase to that of NP.

Overall, for a given timing constraint, the number of pipeline stages leading to the best power/energy efficiency tends to be in the middle of the possible numbers. A similar feature is observed in [41] where the increase of flip-flops gradually reduces the power of a direction detector until the minimum is reached and then, as more flip-flops are added, the power increases.

5.6.3.3 Area versus the Placement Approaches

Comparing the two approaches under the same number of pipeline stages where both are applicable, the overall trend is that the component level approach is the more efficient approach for the timing constraint ranged from tight to modest, as is reflected in the cases from 0.60 ns/1.67 GHz to 1.50 ns/667 MHz. An exception occurs in the case 0.40 ns/2.50 GHz where G4 is better than P4. This is because the target timing constraint of P4 (0.40 ns) is very close to the minimal it can reach (0.38 ns), as is shown in Figure 5.5 (j). When the timing constraint is further loosened, the area cost of the component level increases above that of the gate level approach, as is seen in the cases from 1.50 ns/667 MHz to 8.0 ns/125 MHz. This is explainable since the gate level approach has the flexibility in the placement of registers for further area reduction while the component level approach has no potential for area reduction over the tighter timing constraint cases, as is seen in Figure 5.5 (h) to (j).

5.6.3.4 Power/Energy versus the Placement Approaches

The trend of the power/energy versus the placement approaches does not exactly follow that of the area versus the placement approaches. Similarly, the gate level approach is more efficient in power/energy for the timing constraints from tight to medium. For the further loosened timing constraints, while the gate level approach leads to less area, it consumes more power/energy compared with the component level approach. This means that the less area comes at the cost of compromised power/energy efficiency, as is also found in Section 5.6.1.

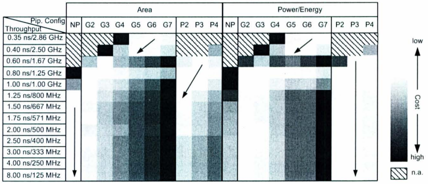


Figure 5.10: Trends of optimal pipeline configurations for the throughput requirements from high to low

5.6.3.5 Energy-wise Costs

The energy-wise costs in Figures 5.7, 5.8 and 5.9 are generated by the candidate implementation with the minimal energy consumption (according to Figure 5.5) running at the clock frequencies lower than its designated frequency. These power/energy costs are lower than the corresponding costs generated by the candidate implementation designated for the clock frequency but result in higher area (e.g., EG2 vs. G2). By comparing between the energy-wise costs, the trends follow those of the regular costs under the loose timing constraints. With the decrease of pipeline stages, the minimal area is achieved by ENP while the minimal power/energy remains at EP3. The component level approach is better than the gate level approach for both area and power/energy.

5.6.3.6 Trends in One Picture

For a more intuitive comprehension of the trends analyzed above, the values shown in Figures 5.7, 5.8 and 5.9 are converted to a gray scale map in Figure 5.10, where a level of gray indicates the cost (the brighter, the lower) of the pipeline configuration compared with others under the timing constraint. For any row in Figure 5.10, the brightest and the darkest correspond to the lowest and the highest costs, respectively, in the corresponding subfigure of Figures 5.7, 5.8 and 5.9, while the other shades in the row are determined to be between them. The energy-wise costs in Figures 5.7, 5.8 and 5.9 are used for the power/energy columns.

Figure 5.10 shows a highly regular variance of area and power/energy costs under the pipeline configurations and timing constraints. The arrows indicate the trends of the most efficient pipeline configurations while the timing constraint varies from tight to loose.

5.6.4 Benefits of Using Pipelined S-Box Implementations

Through the analysis of the candidate implementation results, the benefits of using pipelined S-box implementations with an appropriate pipeline configuration are clearly seen. These benefits can be sorted into three categories: 1) benefits over non-pipelined implementations, 2) benefits over other pipeline configurations, and 3) benefits of providing more performance options/trade-offs.

5.6.4.1 Benefits over Non-Pipelined Implementations

These benefits lie in the timing, area, power and energy for high throughput requirements or lie in power/energy for medium to loose timing constraints. According to our experiment, there is maximum reduction of the critical path delay of 61% for the minimized timing constraint (NP vs. G6 in Figure 5.6). There are maximum reductions of 51% in area and 69% in power/energy for a given tight timing constraint (NP vs. P3 in the case 0.80 ns/1.25 GHz of Figure 5.7 (d)). There is maximum reduction of 30% in power/energy for a medium to loose timing constraint (NP vs. P2 in the case 3.0 ns/333 MHz of Figure 5.8 (d)).

It is worth noticing that, even under the very loose timing constraints, there is a considerable reduction in power/energy of 28% (NP vs. P2 in both cases in Figure 5.9) or 16% (ENP vs. EP3 in both cases in Figure 5.9) with compromised area efficiency. This encourages the application of pipelined S-box implementations in lightweight AES implementations, which usually run at a low clock frequency and have traditionally rarely use pipelined S-box implementations.

5.6.4.2 Benefits over Other Pipeline Configurations

These benefits always exists for an appropriate pipeline configuration that leads to the most efficient implementation. The appropriate pipeline configuration varies with the timing constraints, as is displayed by Figure 5.10.

5.6.4.3 Benefits of Providing More Performance Options/Trade-Offs

The performance options/trade-offs can be significantly increased by using pipelined S-box implementations compared with using only non-pipelined implementations. In this way, besides the most efficient pipeline configuration, other pipeline configurations could also lead to an implementation with desirable performance. For example, in the case 1.50 ns/667 MHz of Figure 5.7 (g), NP and EP3 are the most efficient pipeline configurations in terms of area and power/energy, respectively. As a trade-off between them, P2 costs less area than NP and less power/energy than EP3. Such a combination of area and power/energy of P2 may be preferable for the design with a combined requirement compared to an individual requirement for best area from NP or power/energy from EP3.

5.7 Generality of the Methodology and Results

In this section, we discuss the generality of the methodology and the experimental results presented in this work.

In our methodology, the influence of the pipeline configurations on the performance of S-box implementations is drawn based on the analysis of our experimental results. We believe this is the most straightforward method to have a realistic evaluation of the influence since the experimental implementations are achieved following the same design flow with which realistic implementations are built. All the experimental implementations are benchmarked upon the same technology library, leading to a fair and objective comparison between the pipeline configurations. Although the absolute quantities of the experimental results are technology-specific, the relative comparisons should hold closely if another technology is applied and our analysis and conclusions are mostly based on the relative instead of the absolute values. As well, we also try to mitigate the impact of the technology-specific performance by using gate equivalence as the metric for area.

Another issue related to generality is the gate level placement approach. This approach is synthesis tool dependent since it relies on the retiming function of the tool. However, this dependency should not have a major impact on the general conclusions considering that whatever synthesis tool is used, the retiming function should work towards the similar goal as the one we use and hence the results should not significantly differ.

5.8 Summary

This chapter presents a comprehensive study of pipeline configurations generally applicable to the AES S-box with a composite field structure. In particularly, we investigate an extensive range of pipeline configurations for their influence on the performance of the S-box implementations in terms of timing, area, power and energy. The pipeline configurations consist of no pipeline, a component level pipeline with the number of pipeline stages from 2 to 4 and a gate level pipeline with the number of pipeline stages from 2 to 7. The gate level pipeline utilizes the retiming function of the synthesis tool for the feasible and effective register placement that is compliant with the standard ASIC design flow. Totally 127 S-box implementations with varying pipeline configurations are built with a 90-nm standard cell CMOS technology under a variety of timing constraints.

Based on the performance of these implementations, the influence of the pipeline configurations is discussed with trends that indicate how the appropriate pipeline configurations for certain perspectives of the performance vary with the timing constraints. The trends are found to be highly regularly and explainable. They can be used as the general reference for choosing the appropriate pipeline configurations under a given design requirement in timing. By using the appropriate pipeline configurations, notable performance improvement can be achieved compared with the non-pipelined case. This indicates the merits of applying pipelined S-box implementations for resource-efficient purposes, including lightweight applications. In addition to the appropriate pipeline configurations that lead to the most efficient implementations, it is also shown that other pipeline configurations are able to provide desirable performance trade-offs in S-box implementations.

This work is the first that extensively investigates the pipeline configurations for the AES S-box with a composite field structure and the resulting performance trade-offs. It is also the first work that applies pipelining to AES S-box implementations for resource efficient purposes, as is contrary to the conventional pipelining purpose of speedup. It discloses the benefits of pipelined S-box implementations in terms of resource efficiency. It also introduces the retiming function as a practical and effective register placement approach for the gate level pipelining of S-boxes.

In the next chapter, we look at another perspective of the architecture of AES implementations, the datapath architecture, and investigate the performance improvement and trade-offs provided by different datapath architectures.

Chapter 6

Exploration of Datapath Architectures for Flexible and Efficient Implementation

In this chapter, we present the investigation of the performance of a variety of AES datapath architectures based on the S-boxes with a composite field structure. These architectures are parameterized by a datapath width of 8, 16, 32, 64, or 128 bits and, for the 128-bit width, an unrolling factor of 1, 2, 5 or 10. Through this characterization, the performance trade-offs affected by the architecture parameters are extensively explored. The parameters leading to the best performance are identified. It is found that the 8-bit width datapath, which is conventionally adopted for resource efficient purposes, has the worst energy efficiency and does not result in the minimal peak power among the architectures. As well, the 16, 32 and 64-bit width AES datapath architectures are newly considered or represent improvements over previous work.

6.1 Introduction

The flexibility of the AES algorithm allows parameterizable datapath architectures for hardware implementations. There are two parameters that can specify the datapath architecture of an AES implementation, the datapath width and the unrolling factor. Generally, the possible parameters include the datapath widths of 8, 16, 32, 64 and 128 bits and the unrolling factors of 1, 2, 5 and 10 for the 128-bit width. The fully unrolled pipelined architecture and the fully serialized architecture can be parameterized as an unrolling factor of 10 and a datapath width of 8 bits, respectively. It is obvious that these two architecture parameters can lead to the implementations with the maximum throughput and the minimal area, respectively. However, there are design requirements other than the maximum throughput or the minimal area (e.g., low power/energy or trade-offs between area and throughput). It is unclear which one among the possible architectures can provide superior performance for such design requirements. This motivates this work to characterize the performance of implementations specified by the architecture parameters. We consider the performance in terms of the area, peak power consumption and average energy consumption under a given throughput requirement and there are a variety of throughput requirements considered.

Since there is no standard architecture for given datapath parameters, we consider generic parameterized AES datapath architectures for the characterization and implement them based on the same standard cell CMOS technology so that the characterization results are generalizable. These architectures are designated to perform the AES encryption with 128-bit keys (referred to as AES-E128 in the following). S-boxes with a typical composite field structure are adopted in these architectures. The storage elements in these architectures are all based on registers or shift registers that are composed of only standard cells in the CMOS technology.

A similar work based on FPGA technology is seen in [43]. However, it limits its investigation to the architectures with an unrolling factor of 1, 2, 5 and 10 and only the trade-offs between area and delay are explored. As well, for each architecture, only the implementation synthesized for the maximum throughput is considered. Since different timing constraints for synthesis could lead to quite different implementations with different performance properties, they can not all be realized by synthesizing under the tightest timing constraint for the maximum throughput. The architectures in this work are synthesized with a variety of timing constraints that fit the throughput requirements of a wide range of AES applications, so that a panoramic view of the performance of the implementations affected by the architecture parameters is presented.

Another contribution of the work is the characterization of the novel shift register based 16, 32 and 64-bit width datapath architectures. This provides more performance trade-offs between the 8-bit and the 128-bit width architectures. These architectures are designed for efficiency and generality. The number of clock cycles required to complete an AES round is the minimal for the width, which is 8, 4 and 2, respectively. No specific memory macro is required since all the components in the architectures are composed of standard cells.

6.2 The Datapath Architectures of AES

In this chapter, we consider the datapath architectures that include the datapath widths of 8, 16, 32, 64 and 128 bits and, for the 128-bit width, the unrolling factors of 1, 2, 5 and 10. Correspondingly, we build the generic parameterizable datapath architectures supporting these parameters for AES-E128, which performs the encryption-only operation and has the key size of 128 bits and accordingly 10 rounds.

6.2.1 Common Issues

The common issues related to the datapath architectures characterized in this chapter are described in the following.

6.2.1.1 S-box Structure

All the datapath architectures are based on the S-boxes with a composite field structure from [5]. Although there are other S-box structures, such as such as minimized combinational logic functions from the truth table [18] and decoder-permutation-encoder structure [14], S-box implementations with a composite field structure provide for balanced performance over timing, area and power [18] and are frequently adopted for AES implementations with various architectures, such as in [19] and [20] with a fully unrolled architecture and in [30] and [32] with an 8-bit width datapath architecture. Therefore, we also adopt a composite field structure for the S-box implementations in the architectures. There are a number of composite field structures available, including [5], [6], [7], [8], [9], [10] and [13]. The performance of these composite field structures are similar. We pick the one from [5] since it is a typical composite field structure and the generic architectures characterized in this chapter are expected to show the typical performance that can be provided by the parameterized architectures.

6.2.1.2 Key Expansion

Since we only consider the encryption datapath of AES without the key expansion, for all the architectures, it is assumed that the round keys are fed as inputs to the architectures whenever they are required.

6.2.1.3 Impact of Modes of Operation

The datapath architectures also differ when considering the impact of block cipher modes of operations. We consider the two groups of modes of operations, non-feedback modes (e.g., electronic codebook (ECB) mode and counter (CTR) mode) and feedback modes (e.g., cipher-block chaining (CBC) mode, cipher feedback (CFB) mode and output feedback (OFB) mode). While all of the architectures can work under any mode of operation, the complete datapath architectures with the unrolling factors of 2, 5 and 10 would not be fully utilized when working under feedback modes since only the encryption of one plaintext block can be processed at one time due to the dependency between the current encryption and previous encryption. For this reason, we exclude the situation of the architectures working under feedback modes when comparing performance.

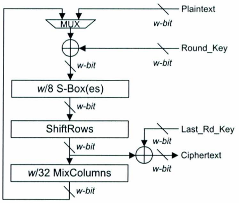


Figure 6.1: Generic model of the partial datapath architectures with width $w \in \{8, 16, 32, 64\}$.

6.2.2 Partial Datapath Architectures

A generic model of the partial datapath architectures is shown in Figure 6.1 where w is equal to the datapath width of a specific architecture and $\oplus$ denotes the bitwise XOR operation for *AddRoundKey*.

Basically, the architectures have a w-bit datapath and on the path, *AddRoundKey*, *SubBytes*, *ShiftRows*, *MixColumns* operations are performed in the sequence. A 128-bit plaintext block is loaded in w-bit pieces in serial, and after the required number of iterations over the architecture, the ciphertext block is loaded out in the same way. The round keys are also loaded in w-bit pieces. The last round key of each encryption is loaded through the specific input *Final_Key_In*, so that the next plaintext can be loaded in while the current ciphertext is being loaded out. This allows the maximum utilization of the architectures without any idle hardware during loading

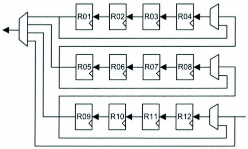


Figure 6.2: Structure of the *ShiftRows* component for the partial datapath architectures with the width of 8 bits.

plaintexts/cipertexts. On average, the datapath architecture with w-bit width can complete the encryption of a plaintext block with $(128/w) \times 10$ clock cycles (i.e., 160, 80, 40, 20 cycles required for the architectures with 8-bit, 16-bit, 32-bit and 64-bit width, respectively).

In addition to the datapath width, the partial datapath architectures differ in the number of S-boxes, the *ShiftRows* component and the *MixColumns* component. The details are presented as follows.

6.2.2.1 S-boxes

Each architecture has $w/8$ S-box(es) in parallel on the datapath. The S-boxes perform the *SubBytes* operation and have a composite field structure from [5].

6.2.2.2 *ShiftRows* Components

Each of the partial datapath architectures has a specific shift register based component for the *ShiftRows* operation. The structures of the components are shown in

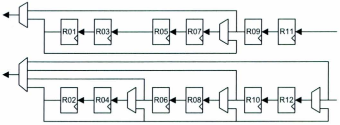


Figure 6.3: Structure of the *ShiftRows* component for the partial datapath architectures with the width of 16 bits.

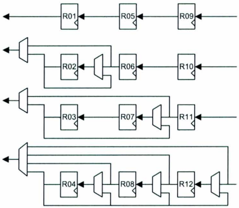


Figure 6.4: Structure of the *ShiftRows* component for the partial datapath architectures with the width of 32 bits.

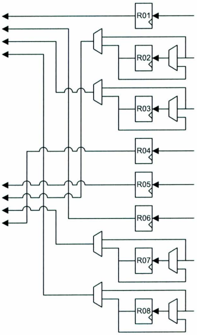


Figure 6.5: Structure of the *ShiftRows* component for the partial datapath architectures with the width of 64 bits.

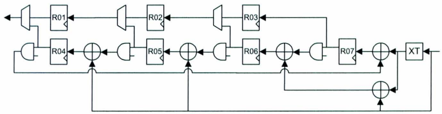


Figure 6.6: Structure of the *MixColumns* component for the partial datapath architectures with the width of 8 bits.

Figures 6.2, 6.3, 6.4 and 6.5. The components are composed of registers and multiplexers. Each path in Figures 6.2, 6.3, 6.4 and 6.5 has the width of 8 bits. The *ShiftRows* component for the 8-bit datapath architecture in Figure 6.2 was proposed in [44]. These components work as shift registers with the multiplexers determining the flow of the data. The components can work without idle cycles by feeding inputs and producing outputs continuously in each clock cycle. For the architectures with the widths of 8, 16, 32 and 64 bits, it takes 16, 8, 4 and 2 cycles, respectively, to complete the *ShiftRows* operation of a *State*. The details of the operation of the components are described in Appendix A.

6.2.2.3 *MixColumns* Components

MixColumns is defined as an operation on 32-bit data. For the datapath architecture with the width of 32 or 64 bits, one or two complete *MixColumns* operations are implemented on the datapath. For those with the width of 8 or 16 bits, *MixColumns* can be partially implemented as 1/4 or 1/2 operation, respectively, and the implementation is reused to complete one *MixColumns* operation. The structures of the

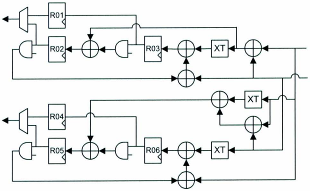


Figure 6.7: Structure of the *MixColumns* component for the partial datapath architectures with the width of 16 bits.

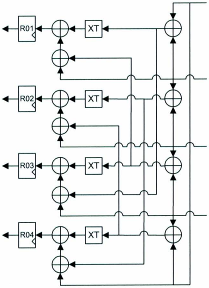


Figure 6.8: Structure of the *MixColumns* component for the partial datapath architectures with the width of 32 bits.

Table 6.1: Comparison of 32-bit AES datapath architectures

	Ours	[26]	[27]	[28]	[29]
Storage Element (in bits)	128	128	256	256	224
Clock Cycles	40	64	40	80	40

components are shown in Figures 6.6, 6.7 and 6.8, respectively.

The components are composed of registers, multiplexers, xtime (XT) components, XOR gates and AND gates (if applicable). The xtime operation is equivalent to the multiplication of the input byte with the hexadecimal value 02 in $GF(2^8)$ modulo $m(x) = x^8 + x^4 + x^3 + x + 1$. It can be implemented with 3 XOR gates, as is shown in [45]. The AND gates are used to bypass the attached XOR gates. Each path in Figures 6.6, 6.7 and 6.8 has the width of 8 bits. The *MixColumns* component for the 8-bit datapath architecture in Figure 6.6 was proposed in [31]. The *MixColumns* component for the 64-bit datapath architecture consists of two copies of that for the 32-bit datapath architecture (Figure 6.8). The 8-bit, 16-bit, 32-bit and 64-bit *MixColumns* components perform the complete *MixColumns* operation on a *State* in 16, 8, 4 and 2 clock cycles, respectively, by feeding inputs and producing outputs continuously in each clock cycle. The details of the operation of the components are described in Appendix B.

6.2.2.4 Novel 16-bit, 32-bit and 64-bit Datapath Architectures

The partial datapath architectures with the width of 16, 32 and 64 bits are novel architectures. Architectures with 16-bit or 64-bit datapath widths have not been discussed in previous literature. The 64-bit architecture requires the minimal amount of storage equivalent to a 128-bit register and the minimal number of clock cycles to complete an encryption (20 cycles) for a 64-bit AES datapath architecture. For the 16-bit architecture, although it is possible to be built with the minimal amount of storage equivalent to a 128-bit register, we found that the overall area can be smaller with 16 more bits of storage (totally equivalent to a 144-bit register) while still using the minimal number of 80 cycles clock cycles to complete the encryption of a plaintext block for a 16-bit AES datapath architecture.

Architectures with a 32-bit datapath width have been investigated in previous work, including [26], [27], [28] and [29]. Since the results of previous work are either based on older ASIC technology than used in this work ([26]) or based on FPGA technology ([27], [28] and [29]), we make a rough comparison between these architectures based on the size of storage and the number of clock cycles to complete the encryption of a plaintxt block, as is shown in Table 6.1.

The storage in an architecture is necessary to hold the updated *States* in an AES datapath and usually takes up a significant amount of the total hardware overhead of the architecture. The minimal size of the storage for an AES datapath architecture is 128 bits. The number of clock cycles per encryption of a block and the critical path delay determine the throughput of the architecture implementation. Considering that the architectures under comparison all have the critical path delay determined by the operation of the round function, the critical paths of the architectures would be similar if they are based on the same technology and, hence, the number of clock cycles becomes the dominant factor determining the throughput.

According to Table 6.1, our 32-bit AES datapath architecture is superior in at

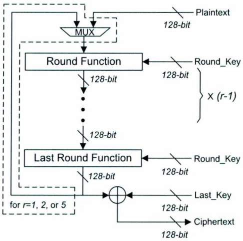


Figure 6.9: Generic model of the complete datapath architectures with the unrolling factor $r \in \{1, 2, 5, 10\}$.

least one of the two metrics compared with the previous works. Among the architectures under comparison in Table 6.1, the storage of [26] and [29] is based on registers while the storage of [27] and [28] is based on memory. The *ShiftRows* operation in the register-based architectures [26] and [29] is not realized as efficiently as in our design so that either more clock cycles or more registers are required. For the memory-based architectures [27] and [28], the *ShiftRows* operation is realized by the appropriate addressing while transferring the *State* bytes between two 16-byte memories, and hence the storage of 256 bits is required. Doubling the number of clock cycles is required for the architecture [28] compared with that of [27] since [28] has separate loops for the *SubBytes* operation and the *MixColumns* operation.

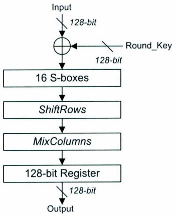


Figure 6.10: Structure for unrolled architectures of the round function.

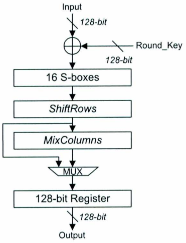


Figure 6.11: Structure for unrolled architectures of the last round function for $r \in \{1, 2, 5\}$.

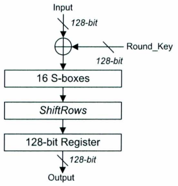


Figure 6.12: Structure for unrolled architectures of the last round function for $r = 10$.

6.2.3 Complete Datapath Architectures

A generic model of the complete datapath architectures is shown in Figure 6.9 where r is equal to the unrolling factor. The complete datapath architecture with the unrolling factor r, $r \in \{1, 2, 5, 10\}$, contains r round functions in series, a 128-bit multiplexer (for $r \in \{1, 2, 5\}$) and a 128-bit bitwise XOR operation, denoted by $\oplus$. To allow for simultaneous processing of r encryptions in the datapath, the round level pipeline is employed, which means there are pipeline registers placed between the hardware of two consecutive rounds.

The round functions are identical except for the last round function. The structure of the round function is shown in Figure 6.10 and the structures of the last round function for $r \in \{1, 2, 5\}$ and for $r = 10$ are shown in Figures 6.11 and 6.12, respectively. The round function in Figure 6.11 is able to work as either the ordinary round function, as in Figure 6.10, or the round function without the *MixColumns* operation for the last round of AES. The last round function in Figure 6.12 only performs the round function without the $MixColumns$ operation.

The $ShiftRows$ components in Figures 6.10, 6.11 and 6.12 are implemented simply as crossover wiring according to the definition of the $ShiftRows$ operation. The $MixColumns$ components in Figures 6.10, 6.11 and 6.12 are implemented with four copies of the 32-bit $MixColumns$ component in Figure 6.8 concatenated in parallel.

For $r \in \{1, 2, 5\}$, the architecture has an iterative loop within which r plaintext encryptions are being processed. The data iterates for the required number before the ciphertext is generated. For $r = 10$, the system processes 10 encryptions simultaneously and the plaintext goes through the datapath once to generate the ciphertext. For the maximum utilization of the architecture, r plaintext blocks can be loaded continuously and processed simultaneously in a pipeline by the r round functions. This is allowed for the cipher using a non-feedback cipher mode, such as counter mode. For maximum utilization, the loading of the following plaintext blocks is paralleled with the unloading of current ciphertext blocks. With the maximum utilization, on average, the architectures with $r \in \{1, 2, 5, 10\}$ take 10 clock cycles to complete the encryption of a plaintext block. Hence, the throughput is determined as r blocks every 10 clock cycles.

6.3 Methodology

In order to characterize the performance of the datapath architectures, all the architectures presented above are implemented with a 90-nm standard cell CMOS technology from STMicroelectronics. The performance of the architectures are estimated based on the synthesis results of the implementations. The performance we consider for an implementation includes area, peak power consumption and average energy consumption. The method to derive the architecture implementations and to estimate the performance are described in the following.

6.3.1 Deriving the Architecture Implementations

In practice, AES implementations are built for applications with various throughput requirements, from high throughput for high speed applications to low throughput for lightweight applications. In order to present the results adapted to a wide range of applications, each datapath architecture is synthesized with a variety of timing constraints to generate the architecture implementations meeting a number of selected throughput requirements that range from high to low.

The timing constraints for a datapath architecture are determined in the way that the implementations are clocked to produce the selected throughputs. The list of the selected throughputs and the corresponding assignments of the timing constraints for each of the architectures are shown in Table 6.2. In Table 6.2, the partial datapath architectures with the width of w bits are denoted as Ww and the complete datapath architectures with the unrolling factor of r are denoted as Ur. Since the minimal realistic critical path delay of the architectures is very close to 1.50 ns under the given technology library. Hence, the timing constraint of 1.50 ns is regarded as the tightest realistic constraint. In Table 6.2, each timing constraint indicates there is an implementation built with the architecture of the row and meeting the throughput

Table 6.2: Assignments of the timing constraints (in ns) for the architectures according to the given throughputs

	85.3 Gbps	42.7 Gbps	17.1 Gbps	8.53 Gbps	4.27 Gbps	2.13 Gbps	1.07 Gbps	533 Mbps	80.0 Mbps	8.00 Mbps	800 kbps
W08	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.50	10.0	100.0	1k
W16	N/A	N/A	N/A	N/A	N/A	N/A	1.50	3.00	20.0	200.0	2k
W32	N/A	N/A	N/A	N/A	N/A	1.50	3.00	6.00	40.0	400.0	4k
W64	N/A	N/A	N/A	N/A	1.50	3.00	6.00	12.0	80.0	800.0	8k
U01	N/A	N/A	N/A	1.50	3.00	6.00	12.0	24.0	160.0	1.6k	16k
U02	N/A	N/A	1.50	3.00	6.00	12.0	24.0	48.0	320.0	3.2k	32k
U05	N/A	1.50	3.75	7.50	15.0	30.0	60.0	120.0	800.0	8k	80k
U10	1.50	3.00	7.50	15.0	30.0	60.0	120.0	240.0	1.6k	16k	160k

of the column clocked at the frequency corresponding to the timing constraint (e.g., the timing constraint of 3 ns corresponds to the clock frequency 333 MHz) and N/A indicates that the corresponding throughput is not achievable for the datapath architecture even under the tightest timing constraint. The architecture implementations are assumed to work with the maximum utilization, i.e., under a non-feedback cipher mode with continuously available plaintext and round keys.

The synthesis and virtual layout process is performed in the topographical mode of Synopsys Design Compiler, as is described in Chapter 3. The synthesis library is the 90-nm CMOS standard cell library from STMicroelectronics with a core voltage of 1.2V and standard threshold voltage.

6.3.2 Evaluation of the Performance

After the architecture implementations are built, their performance in terms of area, peak power and average energy is estimated following the performance evaluation methodology described in Chapter 3. The average power and peak power consumptions are estimated for the implementation running at the clock frequency that produces the corresponding selected throughput. The average energy consumption of an architecture implementation is calculated as the product of its average power consumption, the clock period and the clock cycles per encryption of a plaintext block. Thus, the energy consumption is normalized to be the average energy consumed to encrypt one 128-bit plaintext.

The average power and peak power consumption of the architecture implementations are estimated using PrimeTime PX from Synopsys. The switching activity is obtained from the gate-level simulation of the netlist with 10,000 random 128-bit plaintexts and the corresponding random round keys and control signals. The same set of random plaintexts and round keys is used for the power estimation of all the architecture implementations. PrimeTime PX can also break down the average power of an implementation into the average dynamic power and the average static power or the average powers of the combinational logic and the sequential logic of the circuit. Accordingly, the average energy caused by the dynamic power and the static power or the combinational logic and the sequential logic can be calculated.

6.4 Experimental Results and Analysis

The performance of the implementations of the various architectures are presented and analyzed in this section.

6.4.1 Area

Table 6.3 presents the normalized areas of the architecture implementations. As is expected, the area grows with the datapath width and the unrolling factor. For a given architecture, the area drops with the loosening of the timing constraint until the minimal area is reached and the resulting implementations with looser timing constraints with the same area are actually the same implementation. According to Table 6.3, the most area-efficient implementation is achieved by the architecture with the width of 8 bits (W08), around 1/47 of the size of the datapath with the unrolling factor of 10, under the throughputs achievable by the architecture W08 (from 533 Mbps to 800 kbps).

Table 6.3: Areas of the architecture implementations (normalized to 978 GE)

	85.3 Gbps	42.7 Gbps	17.1 Gbps	8.53 Gbps	4.27 Gbps	2.13 Gbps	1.07 Gbps	533 Mbps	80.0 Mbps	8.00 Mbps	800 kbps
W08	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.12	1	1	1
W16	N/A	N/A	N/A	N/A	N/A	N/A	1.5	1.29	1.28	1.28	1.28
W32	N/A	N/A	N/A	N/A	N/A	2.08	1.85	1.78	1.78	1.78	1.78
W64	N/A	N/A	N/A	N/A	3.78	3	2.97	2.97	2.97	2.97	2.97
U01	N/A	N/A	N/A	7.33	5.48	5.35	5.35	5.35	5.35	5.35	5.35
U02	N/A	N/A	13.31	10.41	10.25	10.25	10.25	10.25	10.25	10.25	10.25
U05	N/A	36.84	27.77	24.72	24.72	24.72	24.72	24.72	24.72	24.72	24.72
U10	60.01	48.1	47.25	47.25	47.25	47.25	47.25	47.25	47.25	47.25	47.25

Table 6.4: Ratios of the area to the maximum throughput of the architectures (normalized to the value of U10)

Architecture	W08	W16	W32	W64	U01	U02	U05	U10
Area/Throughput	2.96	2	1.39	1.26	1.22	1.11	1.23	1

We also determine the area to throughput ratio of the implementation yielding the maximum throughput for each of the architectures and results, normalized to the U10 result, are shown in Table 6.4. For example, the implementation of W32 with the throughput 2.13 Gbps and the implementation of U02 with the throughput 17.1 Gbps are used to derive the corresponding values in the table. In this comparison, U10 is the most efficient architecture in terms of the area cost yielding the one unit of throughput. Roughly, the efficiency becomes worse with the decrease of the unrolling factor or the datapath width. Architecture W08, which leads to the most compact implementations, results in the least efficient architecture and is about 3 times worse than U10 in its area to throughput ratio.

6.4.2 Peak Power Consumption

For the AES implementations targeted at passively powered devices (e.g., contactless smart cards and RFID tags), there is rigorous constraint on the peak power consumption since these devices usually have a very tight budget on power consumption that is shared by all the components on the device. For this reason, the peak power consumption of the architecture implementations is investigated in this section.

Table 6.5: Peak powers of the architecture implementations (normalized to 66.8 μW)

	85.3 Gbps	42.7 Gbps	17.1 Gbps	8.53 Gbps	4.27 Gbps	2.13 Gbps	1.07 Gbps	533 Mbps	80.0 Mbps	8.00 Mbps	800 kbps
W08	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.72	1.46	1.46	1.46
W16	N/A	N/A	N/A	N/A	N/A	N/A	1.47	1.34	1.32	1.32	1.32
W32	N/A	N/A	N/A	N/A	N/A	1.24	1.12	1.08	1.08	1.08	1.08
W64	N/A	N/A	N/A	N/A	1.58	1.07	1	1	1	1	1
U01	N/A	N/A	N/A	3.59	1.94	1.22	1.23	1.23	1.23	1.23	1.23
U02	N/A	N/A	5.8	2.66	2.26	2.26	2.26	2.26	2.26	2.26	2.26
U05	N/A	17.84	9.35	5.85	5.85	5.85	5.85	5.85	5.85	5.85	5.85
U10	19.87	13.16	11.39	11.39	11.39	11.39	11.39	11.39	11.39	11.39	11.39

Table 6.5 presents normalized peak power of all the architecture implementations. For a given architecture implementation, the peak power would not vary with the clock frequency, as is shown in Table 6.5. Under the same given throughput, the architecture resulting in the minimal peak power is W64. It is obvious that the peak power of an implementation is related to its area. This is the reason that the increase of the unrolling factor beyond U01 increases the peak power. However, for the partial datapath architectures, the decrease of the datapath width leads to the increase of the peak power according to Table 6.5. This reflects the fact that there are more intense instantaneous switching activities incurred by the partial datapath architecture with a smaller datapath width. Especially for W08 and W16, they have more registers than other partial datapath architectures and a register consumes more power than other CMOS cells and is updated every clock cycle. Between W64 and U01, W64 also leads to less instantaneous switching activities and achieves the overall lower peak power.

6.4.3 Average Energy Consumption

Energy consumption is usually a crucial constraint for battery-powered devices since the capacity of the battery is limited. It is preferred to perform as many tasks as possible under a given capacity of the battery. The average energy required to encrypt a 128-bit plaintext block for each architecture implementation is shown in Table 6.6.

It can be seen in Table 6.6 that, for any architecture, loosening the timing constraint would lead to the implementation with less energy consumption when working for the designated throughput until the minimal is reached. Past this point, more

Table 6.6: Average energy for the encryption of 128-bit plaintext of the architecture implementations (normalized to 0.73 nJ)

	85.3 Gbps	42.7 Gbps	17.1 Gbps	8.53 Gbps	4.27 Gbps	2.13 Gbps	1.07 Gbps	533 Mbps	80.0 Mbps	8.00 Mbps	800 kbps
W08	N/A	N/A	N/A	N/A	N/A	N/A	N/A	2.4	2.09	2.24	3.92
W16	N/A	N/A	N/A	N/A	N/A	N/A	1.57	1.33	1.54	1.75	3.57
W32	N/A	N/A	N/A	N/A	N/A	1.28	1.23	1.22	1.24	1.53	4.4
W64	N/A	N/A	N/A	N/A	1.38	1.26	1.25	1.25	1.3	1.78	6.55
U01	N/A	N/A	N/A	1.4	1.34	1.34	1.32	1.33	1.41	2.27	10.9
U02	N/A	N/A	1.53	1.3	1.25	1.25	1.26	1.27	1.43	3.1	19.75
U05	N/A	1.8	1.37	1.33	1.34	1.34	1.37	1.4	1.81	6.09	48.95
U10	1.16	1.01	1	1	1.01	1.03	1.06	1.13	1.9	10.05	91.49

Table 6.7: Average energy incurred due to static power for the encryption of 128-bit plaintext block of the architecture implementations (normalized to 1.1 pJ)

	85.3 Gbps	42.7 Gbps	17.1 Gbps	8.53 Gbps	4.27 Gbps	2.13 Gbps	1.07 Gbps	533 Mbps	80.0 Mbps	8.00 Mbps	800 kbps
W08	N/A	N/A	N/A	N/A	N/A	N/A	N/A	3	12.64	113	1222.82
W16	N/A	N/A	N/A	N/A	N/A	N/A	2.4	2	15.18	152.09	1354.82
W32	N/A	N/A	N/A	N/A	N/A	1.64	2.18	3.09	21.55	210.45	2105.91
W64	N/A	N/A	N/A	N/A	3.09	6.73	2.64	5.27	35.55	357	3510.45
U01	N/A	N/A	N/A	2.36	2.45	1.73	4.91	9.27	63.09	630.36	6342.18
U02	N/A	N/A	1.73	1.27	2.64	5.36	9.18	18.45	123.27	1227.27	12240
U05	N/A	2.45	3.27	1.73	7.18	9.45	23.64	47.09	314.45	3149.36	31493
U10	1.64	1	3	5.73	12.55	23.45	44.64	89.73	598.27	5988.82	59850.64

energy is consumed when the implementations work for lower throughputs. For these cases, the increase of energy consumption is incurred by the static power of the circuit and is independent of the switching activity but related to the duration the circuit is powered on.

The average energy incurred by the static power of the architecture implementations is shown in Table 6.7. It can be seen in Table 6.7 that the energy consumption due to static power rises dramatically with the drop of the throughput (i.e., the slowing down of clock frequency). This indicates that the highest energy efficiency of the implementation can only be reached with the appropriate clock frequency under which the total energy consumption (the energy by both the dynamic power and the static power) is minimal, such as 80.0 Mbps for W08, 1.07 Gbps for U01 and 17.1 Gbps for U10. Conventionally, AES implementations targeted at lightweight applications are usually made to work at an extremely low clock frequency. According to the above analysis, this, in fact, does not necessarily save energy but may consume more due to static power.

Comparing the implementations with the lowest average energy consumption from each of the architectures, it can be seen that the U10 implementation with the throughput of 17.1 Gbps or 8.53 Gbps is the lowest for U10 while the W08 implementation with the throughput of 80.0 Mbps is the lowest for W08, which is about 2 times that of U10. By comparing the average energy consumption of the implementations for a given throughput in Table 6.6, U10 remains as the most energy efficient architecture for a wide range of the throughputs, from 85.33 Gbps to 533 Mbps. For lower throughputs, since the energy incurred by the static power gradually becomes dominant in the total energy, the architecture implementations with smaller area become more energy efficient. For example, the W08 architecture is substantially more energy efficient than U10 at 800 kbps.

For more in-depth analysis of the energy consumption of the architecture implementations, we also break down the energy incurred by the dynamic power into the energy used by the combinational logic and the energy used by the registers in the implementations. We have found that there is significant difference in the energy caused by the dynamic power of the registers, as is shown in Table 6.8. It can be seen in Table 6.8 that the energy consumption of the registers of the partial datapath architectures is approximately proportional to the clock cycles required to complete the encryption of a 128-bit plaintext block, which are 160, 80, 40 and 20 cycles for W08, W16, W32 and W64, respectively. This is reasonable considering that the partial datapath architectures have a similar number of registers and the registers are updated approximately 160, 80, 40 and 20 times to complete the encryption, respectively. On the other hand, for the complete datapath architectures, they all need to update the similar number of registers 10 times to complete one encryption. Therefore, these architectures have relatively similar energy consumption of the registers and that is about 1/16, 1/8, 1/4 and 1/2 of the energy for W08, W16, W32 and W64, respectively. This finding inspired us to investigate the further reduction of the energy of the complete datapath architectures by removing the registers between the consecutive round functions (excluding the architecture U01). However, our experimental results show that these architectures consume significantly more energy after the registers are removed due to the increment of energy caused by the severe increase of glitching. Roughly, U02, U05 and U10 consume 2, 5 and 10 times more energy after the registers are removed.

Table 6.8: Average energy incurred due to dynamic power of the registers for the encryption of 128-bit plaintext of the architecture implementations (normalized to 34.2 pJ)

	85.3 Gbps	42.7 Gbps	17.1 Gbps	8.53 Gbps	4.27 Gbps	2.13 Gbps	1.07 Gbps	533 Mbps	80.0 Mbps	8.00 Mbps	800 kbps
W08	N/A	N/A	N/A	N/A	N/A	N/A	N/A	18.75	18.45	18.45	18.45
W16	N/A	N/A	N/A	N/A	N/A	N/A	9.13	9.26	8.8	8.8	8.8
W32	N/A	N/A	N/A	N/A	N/A	4.59	4.46	4.02	4.02	4.02	4.02
W64	N/A	N/A	N/A	N/A	2.97	3.15	2.39	2.39	2.39	2.39	2.39
U01	N/A	N/A	N/A	1.55	1.68	1.03	1.03	1.03	1.03	1.03	1.03
U02	N/A	N/A	1.52	1.8	1.14	1.14	1.14	1.14	1.14	1.14	1.14
U05	N/A	1.91	1.43	1.35	1.35	1.35	1.35	1.35	1.35	1.35	1.35
U10	1.49	1.76	1	1	1	1	1	1	1	1	1

Table 6.9: Overall resource cost of the architecture implementations (normalized to the value of W32 under 2.13 Gbps)

	85.3 Gbps	42.7 Gbps	17.1 Gbps	8.53 Gbps	4.27 Gbps	2.13 Gbps	1.07 Gbps	533 Mbps	80.0 Mbps	8.00 Mbps	800 kbps
W08	N/A	N/A	N/A	N/A	N/A	N/A	N/A	5.57	24.79	265.94	4648.7
W16	N/A	N/A	N/A	N/A	N/A	N/A	2.1	2.76	21.27	241.18	4916.7
W32	N/A	N/A	N/A	N/A	N/A	1	1.55	2.83	19.37	238.09	6839.48
W64	N/A	N/A	N/A	N/A	1.26	1.22	2.25	4.49	31.15	428.43	15751.6
U01	N/A	N/A	N/A	2.79	2.17	2.67	5.29	10.55	74.99	1206.91	58082.31
U02	N/A	N/A	4.5	2.73	4.4	8.82	17.79	35.71	268.98	5825.8	371084.89
U05	N/A	17.92	13.5	14.64	29.45	58.97	120.27	244.96	2114.7	71348.51	5732672.21
U10	10.48	9.72	20.42	40.99	82.91	168.2	348.15	735.52	8277.6	437831.35	39855107.53

6.4.4 Overall Resource Cost

In the previous sections, we have evaluated the cost of the architecture implementations for each of the performance perspectives separately. In this section, we evaluate the overall resource cost (ORC) of the architecture implementations by combining their performance in area, peak power consumption, average energy consumption and throughput, as

$$ORC = \frac{Area \times Peak\ Power \times Energy}{Throughput}.$$

This metric reflects the combined cost required to yield the unit throughput and the lower the cost, the higher the efficiency. The overall resource cost of the architecture implementations is shown in Table 6.9.

According to Table 6.9, the architecture implementation W32 under the throughput 2.13 Gbps provides the lowest overall resource cost and the implementations close to that one also have low cost, such as W64 under 2.13 Gbps, W16 under 1.07 Gbps and W08 under 1.07 Gbps. It can also be seen from Table 6.9, when the throughput decreases, the overall resource cost increases dramatically.

6.5 Summary

In this chapter, the performance of parameterizable AES datapath architectures are benchmarked based on a 90-nm standard cell CMOS technology in terms of area, peak power consumption and average energy consumption. For this purpose, generic and representative datapath architectures are built with the architecture parameters including the datapath widths of 8, 16, 32 and 64 bits and the unrolling factors of

1, 2, 5 and 10 for the datapath width of 128 bits. Among these architectures, the 16-bit and 64-bit width architectures have not been discussed before in the literature and the 32-bit width architecture is a novel architecture with the benefit of reduced storage and/or fewer clock cycles compared to previous work.

For each of the architectures, a number of timing constraints are applied to derive the architecture implementations fit for different throughput requirements. The quantitive performance of the architecture implementations are presented, compared and analyzed. The most efficient architecture implementation in area, peak power and energy, as well as in the overall resource cost are identified. The performance trade-offs over a range of architecture parameter values are disclosed. In contrast to conventional belief, the most compact architecture implementation with the 8-bit width does not help to minimize, but actually increases, the energy consumption even running at a low clock frequency. As well, compact implementations do not result in the minimal peak power.

Since the datapath architecture parameter values examined in this work cover an extensive range of the possible values and the architecture implementations are fairly compared based on the same standard cell CMOS technology, the results from this chapter are generalizable and scalable to other technologies. Therefore, this work can serve as a general reference for resource-efficient and flexible implementation of the AES datapath.

In the next chapter, we combine the most energy efficient S-box pipeline configuration from Chapter 5 and datapath architecture identified in this chapter, in order to demonstrate the overall effectiveness of the research results presented in the two chapters.

Chapter 7

Demonstration of Combined Effects for Energy Efficiency

In this chapter, we demonstrate the improvement in energy efficiency of the AES datapath implementation by combining the appropriate pipeline configuration and datapath architecture that are identified in Chapters 5 and 6, respectively. The result shows significant reduction in energy consumption for the datapath implementation with pipelined S-boxes compared with the datapath implementation with non-pipelined S-boxes.

7.1 Introduction

The results in Chapters 5 and 6 show the performance improvements and performance trade-offs under a variety of pipeline configurations for the S-box and architectures for the datapath. In terms of the improvements in resource efficiency, the combination of the appropriate pipeline configuration and the appropriate datapath architecture would further improve the resource efficiency. In the following of this section, we demonstrate the combined effect in resource efficiency by applying the energy-efficient pipeline configuration from Chapter 5 into the energy-efficient datapath architecture implementation from Chapter 6.

Specifically, we apply the 2-stage and 3-stage component level pipeline configurations to the 128-bit datapath architecture with the unrolling factor of 10. According to Chapter 5, the component level 2-stage and 3-stage pipeline are the two most energy-efficient configurations for the timing constraints where they are applicable. According to Chapter 6, the datapath architecture with the unrolling factor of 10 achieves the lowest energy consumption among the datapath architectures. We will build the implementations with the combined pipeline configurations and the datapath architectures and compare the performance with the implementations of the non-pipelined S-box datapath architecture with the unrolling factor of 10 and the datapath architecture with 8-bit width. The 8-bit datapath architecture is conventionally adopted for low power/energy implementation of AES. These implementations do not contain key expansion and it is assumed that the round keys are fed as inputs to the implementations whenever they are required.

7.2 The Methodology

The methodology for implementing the combined architectures and evaluating the performance follows that used in Chapters 5 and 6. In the following, the datapath architecture with the unrolling factor of 10 and 2-stage component level pipelined S-boxes is denoted as U10P2 and the datapath architecture with the unrolling factor of 10 and 3-stage component level pipelined S-boxes is denoted as U10P3. The denotations of other datapath architectures follows from Chapter 6.

The list of the selected throughputs and the corresponding assignments of the timing constraints for the combined architectures is shown in Table 7.1. The selected throughputs include those used in Chapter 6 and some higher throughputs which are achievable for the architectures due to the pipeline. It should be noted that, after introducing the pipelined S-boxes in the datapath architectures, the implementation can still produce a 128-bit output every clock cycle.

7.3 Results and Analysis

The normalized area, peak power and average energy of the implementations of the combined architectures are shown in Tables 7.2, 7.3 and 7.4, respectively. For comparison, the performance of the datapath path architectures W08 and U10 are also presented in the tables.

It can be seen that the combined architectures U10P2 and U10P3 can lead to lower area than the datapath architecture U10 under the throughput 85.3 Gbps in Table 7.2. Also, the combined architecture U10P3 has more pipeline stages in the S-boxes but can lead to lower area than U10P2 under the throughput 171 Gbps.

There is significant reduction in energy by using the combined architectures. Considering the minimal energy consumption for each of the architectures under comparison (i.e., W08 under 10.0 Mbps, U10 under 2.13 Gbps, U10P2 under 5.33 or 2.13 Gbps and U10P3 under 5.33 Gbps) in Table 7.4, the combined architectures U10P2 and U10P3 can save maximally around 40% and 50% energy, respectively, compared

Table 7.1: Assignments of the timing constraints (in ns) for the architectures under comparison according to the given throughputs

	256 Gbps	171 Gbps	85.3 Gbps	42.7 Gbps	17.1 Gbps	8.53 Gbps	4.27 Gbps	2.13 Gbps	1.07 Gbps	533 Mbps	80.0 Mbps	8.00 Mbps	800 Kbps
W08	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.50	10.0	100.0	1k
U10	N/A	N/A	1.50	3.00	7.50	15.0	30.0	60.0	120.0	240.0	1.6k	16k	160k
U10P2	N/A	0.75	1.50	3.00	7.50	15.0	30.0	60.0	120.0	240.0	1.6k	16k	160k
U10P3	0.50	0.75	1.50	3.00	7.50	15.0	30.0	60.0	120.0	240.0	1.6k	16k	160k

Table 7.2: Areas of the combined architecture and selected datapath architecture implementations (normalized to 978 GE)

	256 Gbps	171 Gbps	85.3 Gbps	42.7 Gbps	17.1 Gbps	8.53 Gbps	4.27 Gbps	2.13 Gbps	1.07 Gbps	533 Mbps	80.0 Mbps	8.00 Mbps	800 Kbps
W08	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.12	1	1	1
U10	N/A	N/A	60.01	48.1	47.25	47.25	47.25	47.25	47.25	47.25	47.25	47.25	47.25
U10P2	N/A	71.38	55.95	54.83	54.83	54.83	54.83	54.83	54.83	54.83	54.83	54.83	54.83
U10P3	96.94	61.95	58.79	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65	58.65

Table 7.3: Peak powers of the combined architecture and selected datapath architecture implementations (normalized to 66.8 μW)

	256 Gbps	171 Gbps	85.3 Gbps	42.7 Gbps	17.1 Gbps	8.53 Gbps	4.27 Gbps	2.13 Gbps	1.07 Gbps	533 Mbps	80.0 Mbps	8.00 Mbps	800 Kbps
W08	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.18	1	1	1
U10	N/A	N/A	13.57	8.99	7.78	7.78	7.78	7.78	7.78	7.78	7.78	7.78	7.78
U10P2	N/A	27.4	17.16	15.81	15.81	15.81	15.81	15.81	15.81	15.81	15.81	15.81	15.81
U10P3	54.4	25.87	20.33	19.36	19.36	19.36	19.36	19.36	19.36	19.36	19.36	19.36	19.36

Table 7.4: Average energy for the encryption of 128-bit plaintext of the combined architecture and selected datapath architecture implementations (normalized to 0.35 nJ)

	256 Gbps	171 Gbps	85.3 Gbps	42.7 Gbps	17.1 Gbps	8.53 Gbps	4.27 Gbps	2.13 Gbps	1.07 Gbps	533 Mbps	80.0 Mbps	8.00 Mbps	800 Kbps
W08	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	4.94	4.32	4.63	8.09
U10	N/A	N/A	2.39	2.09	2.06	2.07	2.09	2.13	2.19	2.33	3.92	20.74	188.82
U10P2	N/A	1.68	1.28	1.23	1.23	1.24	1.26	1.3	1.38	1.53	3.27	21.65	205.62
U10P3	2.24	1.11	1.01	1	1.01	1.02	1.04	1.08	1.16	1.33	3.22	23.19	222.87

with the most energy efficient architecture U10 found in Chapter 6. Compared with the datapath architecture W08 which is regarded as the most lightweight implementation architecture for AES, the combined architectures U10P2 and U10P3 require as low as around 1/5 of the minimal energy consumed by W08. The energy reduction by the combined architectures U10P2 and U10P3 compared with U10 comes from the energy reduction by using the component level 2-stage and 3-stage pipelined S-boxes, as is shown in Chapter 5. U10P3 can consumer less energy than U10P2 because the component level 3-stage pipelined S-box has lower energy consumption than the component level 2-stage pipelined S-box as seen in Chapter 5.

These energy reductions come at the cost of the increased peak power. As is seen in Table 7.3, W08 remains as the architecture with the minimal peak power and compared with U10, U10P2 and U10P3 under the minimal energy consumption, the peak power of W08 is about 1/8, 1/16 and 1/20 of them, respectively.

Based on above analysis, it is clear that, for solely low energy purpose, the architecture U10P3 should be adopted and the implementation should be constrained and run under a high (but not the highest) throughput (e.g., between 85.3 Gbps and 4.27 Gbps). The architecture U10P2 leads to the implementations with the similar scenario but has around 7% decrease in area (54.83 versus 58.65 in Table 7.2) at the cost of around 23% increase in energy (1.23 versus 1 in Table 7.4). In terms of area and peak power, the architectures W08 and W64 identified in Chapter 6 remain as the most efficient solutions, respectively.

The significant energy reduction found in this section indicates that those AES implementation architectures that employs the minimal datapath width for low power purpose (characterized as W08 in this dissertation, such as [31]) actually consume much more energy than those architectures built for high throughput purpose (characterized as U10, U10P2 and U10P3, such as [20]).

7.4 Summary

In this chapter, we demonstrate the combined effect in improving energy efficiency by the appropriately combined S-box pipeline configuration and datapath architecture. We combine the component level 2-stage and 3-stage pipelined S-boxes with the datapath architecture with the unrolling factor of 10. The performance evaluation of the implementations of the combined architectures shows there is significant energy reduction achieved by the combined architectures compared with the most energy efficient datapath architecture identified in Chapter 6 or the most compact datapath architecture conventionally used for low resource purposes.

Chapter 8

Design of a Lightweight Block Cipher PUFFIN2

In previous chapters, we have focused on analyzing the implementation of AES. As the alternative of AES, many others have proposed lightweight block cipher algorithms targeted at low complexity implementation. In this chapter, we present a block cipher, PUFFIN2, which is designed to be used with applications requiring very low circuit area. PUFFIN2 is designed to be implemented exclusively with CMOS technologies and in a serialized architecture, so that the maximum reuse of hardware components is achieved resulting in a very compact implementation. PUFFIN2 has a block size of 64 bits and a key size of 80 bits. Compared with a serialized implementation of cipher PRESENT, which has the same block size and key size and is claimed as the smallest practical block cipher implementation to date, our cipher has 16% fewer gates using the same CMOS technology. Further, PUFFIN2 inherently supports both encryption and decryption while the serialized PRESENT is an encryption-only implementation.

The content of this chapter is also presented in [46].

8.1 Introduction

Lightweight applications usually refer to the applications with extremely constrained requirements on cost (complexity), power and/or energy consumption, such as RFID tags and sensor networks. The block ciphers that are targeted at those applications are usually called lightweight block ciphers. Although there have been plenty of efforts put into the investigation of the implementation of AES with low complexity, the complexity inherent in the algorithm imposes a lower bound on the complexity. Most of the efforts on the efficient implementation of AES are actually the exploration of the trade-off between area and delay where the sacrifice of one major aspect of the performance is unavoidable.

Lightweight block cipher design is one of the recent trends in symmetric key cryptography. There are many attempts made to investigate the potential of designing lightweight block ciphers that lead to compact implementations without significant compromise of the delay. Lightweight block cipher algorithms include PRESENT [47][48], mCRYPTON [49], ICEBERG [37], HIGHT [50], SEA [51], PUFFIN [52], KATAN, KTANTAN [53], MIBS[54] and LBlock [55].

It is well known that an efficient method to minimize hardware area is to reuse the single piece of a hardware component for multiple times instead of replicating identical pieces for simultaneous operation. This hardware reduction method, also known as a serialized architecture, is well suited to block ciphers which usually involve cryptographic components consisting of identical function blocks, e.g. non-linear substitution layers generally consist of identical S-boxes. Although this reduction of hardware area comes at the penalty of increased execution time, the compromised timing performance is still acceptable for many applications at which lightweight block ciphers are targeted. The serialized architecture is firstly exploited and applied to PRESENT in [48] and this is the smallest known implementation of a practical block cipher. In the reminder of this chapter, this implementation is called serialized PRESENT.

PUFFIN2 is a block cipher named after its predecessor PUFFIN and designed to be implemented exclusively with a serialized architecture. The differences between PUFFIN and PUFFIN2 lie in the number of rounds, function order in a round and the key schedule which is fully redesigned for PUFFIN2 in order to perform it with the datapath. Both PUFFIN and PUFFIN2 have the capability of both encryption and decryption. In the next section it will be shown that the datapath of PUFFIN2 is exactly the same for encryption and decryption, so there is no hardware overhead to accommodate the difference between encryption and decryption operations.

PUFFIN2 is more efficient than PRESENT for hardware implementation with serialized architecture. PUFFIN2 has the same block size and key size as PRESENT, 64-bit and 80-bit, respectively. Compared with the widely adopted 128-bit key size and 128-bit block size of AES, an 80-bit key size with a 64-bit block size can result in a compact implementation and still provide sufficient security for typical low cost smart devices. Based on our ASIC implementation experiments, PUFFIN2 is realized with 1083 gates which is 16% less than serialized PRESENT with 1296 gates. For those block ciphers appeared after PUFFIN2, including KATAN, KTANTAN [53], MIBS[54] and LBlock [55], they are very similar or even larger area compared with

PRESENT according to the comparison in these works. For this reason, PRESENT is adopted as the reference for comparison in this chapter.

8.2 Cipher Specification

A block cipher is generally constructed from two parts: datapath and key schedule. The datapath is in the form of a product cipher from Shannon's theory [56] where a number of identical or similar functions are concatenated to multiply the security strength of the block cipher. The cipher is composed of a number of rounds, where the operations within a round are referred to as a round function. Each round function takes the output from the previous round (or the plaintext for the first round function) and also receives a set of information (referred as to a round key or round key) from the key schedule, and generates the input for the following round (or the ciphertext for the last round function). Confusion and diffusion are two basic techniques introduced by Shannon [56] to obscure plaintext into ciphertext. Confusion complicates the relationship between the plaintext, the ciphertext and the key, while diffusion spreads the influence of confusion.

8.2.1 Overall Structure

The proposed block cipher, PUFFIN2, adopts a simple involutional Substitution Permutation Network (SPN) [57] with a data block size of 64 bits and key size of 80 bits and consists of 34 rounds. Encryption and decryption processes are identical so the same datapath can be used for both processes. The key schedule of the cipher generates 64-bit round keys for each round on-the-fly (that is, in parallel to the processing of the cipher data).

In a block cipher using an SPN structure, confusion and diffusion are performed with the substitution layer and the transposition or permutation layer in a round function, and the security is enhanced by concatenating the round functions. Compared with the classical Feistel structure [58] where only half of the data block is processed in parallel, the entire data block is processed at one time for the substitution and permutation operations of a round function. An involutional structure allows the cipher to have identical encryption and decryption processes, which is one of the design goals of PUFFIN2. In general, a block cipher using an SPN structure has different datapaths for encryption and decryption because the reverse of the encryption process, which is equal to the decryption process, is usually not designed to be the same as the forward encryption process. In PUFFIN2, we design the same forward and reverse processes for the encryption by adopting the involutional components in the cipher whose forward operations used for encryption can be the same as the reverse operations used for decryption. In this way, there is no seperate encryption datapath and decryption datapath required in the implementation, although encryption and decryption cannot be done simultaneously since they share the same datapath.

Table 8.1: S-box mapping of PUFFIN2 (in hexadecimal)

Input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output	D	7	3	2	9	A	C	1	F	4	5	E	6	0	B	8

Table 8.2: 64-bit Permutation of PUFFIN2

	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
R_0	13	2	60	50	51	27	10	36
R_1	25	7	32	61	1	49	47	19
R_2	34	53	16	22	57	20	48	41
R_3	9	52	6	31	62	30	28	11
R_4	37	17	58	8	33	44	46	59
R_5	24	55	63	38	56	39	15	23
R_6	14	4	5	26	18	54	42	45
R_7	21	35	40	3	12	29	43	64

8.2.2 Basic Components

Each round function of PUFFIN2 consists of 3 layers, a nonlinear substitution layer S, a key addition layer A and a permutation layer P. The nonlinear substitution layer S is composed of 16 identical 4×4 S-boxes, which are the same as the S-boxes used in PUFFIN and the S-box mapping is shown in Table 8.1. 4×4 S-boxes (which are small compared to the 8×8 S-boxes of AES) are often found in lightweight block ciphers because their implementations are compact and their comparative weakness in security strength can be compensated by an increased number of rounds. The key addition layer A performs a bitwise XOR with the 64-bit data block and the 64-bit round key provided by the key schedule. The permutation layer P is a bit transposition of the 64-bit data block.

The permutation scheme of PUFFIN2 is borrowed from the 64-bit data block permutation of PUFFIN which fulfills the criterion that no two outputs of a 4×4 S-box are connected to the same S-box in the next round. The permutation scheme is given in Table 8.2. According to this table, the $(8 \times m + n + 1)$-th input is mapped to the N-th output where N is the value located at C_n and R_m.

As can be seen from Tables 8.1 and 8.2, both the S-box mapping and the permutation are involutional. According to the completeness property of cryptography introduced in [59], the cipher with a tree-structured SPN and a specifically designed permutation can achieve the completeness property within the fewest rounds, which would be 3 for 64-bit cipher with 4×4 S-boxes [59]. In PUFFIN as well as PUFFIN2, due to the requirement of involutional permutation, a tree-structured SPN can not be adopted to achieve the completeness property in only 3 rounds. However, it can be shown that the completeness property can still be reached after five rounds of PUFFIN and PUFFIN2 with the S-box mapping and permutation presented above [60].

8.2.3 Encryption and Decryption Process

The encryption and decryption processes are shown in Figure 8.1, where K_r denotes the r-th round key and $K'_r = P(K_r)$. The whole process consists of 34 rounds plus an extra substitution layer. The explanation of selecting 34 as the number of rounds is given in the next section. The extra substitution layer is required to form identical encryption and decryption processes. For each round of the encryption/decryption process, the 64-bit input data goes through the substitution layer S, the permutation layer P and then adds with the round key to generate the input of the next round.

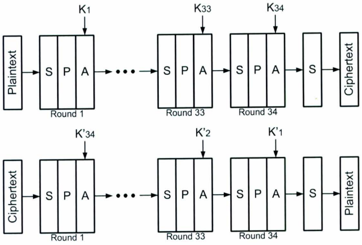


Figure 8.1: Block diagram of the encryption (top) and decryption (bottom) processes

According to Figure 8.1, the encryption process of PUFFIN2 can be represented as follows:

$$\alpha_{34}[K_1, K_2, ...K_{34}] = O_{r=1}^{34}(\mathrm{S} \circ \mathrm{P} \circ \mathrm{A}_{Kr}) \circ \mathrm{S}. \tag{8.1}$$

In the above expressions, the notation $\circ$ means the concatenation of the basic operation in one round such as substitution S and permutation P. The notation O is used to represent the concatenation of 34 rounds of operation of (S $\circ$ P $\circ$ A). Decryption should be as follows:

$$\begin{aligned}\alpha_{34}^{-1}[K_1, K_2, ...K_{34}] &= \mathrm{S} \circ O_{r=34}^{1}(\mathrm{A}_{Kr} \circ \mathrm{P} \circ \mathrm{S}) &(8.2)\\ &= O_{r=34}^{1}(\mathrm{S} \circ \mathrm{A}_{Kr} \circ \mathrm{P}) \circ \mathrm{S}. &(8.3)\end{aligned}$$

Because the substitution layer S and the permutation layer P are involutional, we can have the following relationship:

$$\mathrm{A}_{Kr} \circ \mathrm{P} \equiv \mathrm{P} \circ \mathrm{A}_{P(Kr)}, \tag{8.4}$$

and therefore, we can obtain the following:

$$\alpha_{34}^{-1}[K_1, K_2, ...K_{34}] = O_{r=34}^{1}(\mathrm{S} \circ \mathrm{P} \circ \mathrm{A}_{P(Kr)}) \circ \mathrm{S}. \tag{8.5}$$

The above expression is consistent with the decryption process shown in Figure 8.1, which means the decryption process is similar in form to the encryption process. In the decryption process, the round keys used in encryption are permuted with P and applied in the reverse order.

Table 8.3: Description of the components of the key schedule

Component	Function
S80	Substitution of the 80 bits
PL64	Permutation of the left 64 bits
PR64	Permutation of the right 64 bits
L64	Selection of the left 64 bits
R64	Selection of the right 64 bits

Table 8.4: Round distribution of PL64, PR64, L64 and R64

Round #	Permutation	Selection
r=1,2,33,34	PL64	L64
r=3,4,31,32	PR64	R64
$r = 5 + 4n$,$0 \leq n \leq 6$	PR64	R64
$r = 6 + 4n$,$0 \leq n \leq 6$	PR64	R64
$r = 7 + 4n$,$0 \leq n \leq 5$	PL64	L64
$r = 8 + 4n$,$0 \leq n \leq 5$	PL64	L64

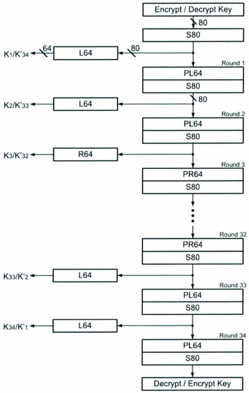


Figure 8.2: Block diagram of the key schedule

8.2.4 Key Schedule

The key schedule of PUFFIN2 operates on an 80-bit key and generates a 64-bit round key for each round on-the-fly. The components used by the key schedule are listed in Table 8.3 and the key schedule is demonstrated in Figure 8.2. The key schedule consists of 34 round functions plus an extra substitution layer at the beginning. Each of the round functions is comprised of a permutation layer PL64 or PR64 and a substitution layer S80. PL64/PR64 permutes the left/right 64 bits of the 80-bit round input and then S80 performs the substitution on the 80 bits of key data. Depending on the selection component (L64 or R64), each 64-bit round key is generated by taking the left or right 64 bits of the 80-bit intermediate value that feeds to the corresponding round function. The detailed distribution of PL64 and PR64 along with L64 and R64 is shown in Table 8.4. The irregular distribution of PL64 and PR64 for each round is intended to prevent related-key attacks and will be discussed in Section 8.3.

In order to maximize hardware resource reuse to achieve a compact implementation, the substitution component S80 is designed to consist of 4×4 S-boxes that are the same as the S-box used in the encryption and decryption processes, and the 64-bit permutation mapping in PL64 and PR64 is also the same mapping as in the permutation layer in the encryption and decryption process.

The key schedule of PUFFIN2 is designed to be involutional to fulfill the design goal of a full involutional block cipher. The involutional property is achieved through the following measures. In the first place, all basic components in the key schedule are involutional. Secondly, the distribution of PL64 and PR64 along with L64 and P64 is symmetric, and in order to achieve this, the round number of the key schedule has to be a number that is double of an odd number and consequently 34 is selected as the round number of the key schedule as well as the encryption and decryption processes. As will be noted in Section 8.3, 34 rounds are also adequate to achieve an appropriate level of security. Thirdly, there is an extra substitution layer S80 at the beginning of the key schedule which makes the forward path of round key generation identical to its backward path. The PL64 and S80 in the last round (round 34) of the key schedule are only useful to compute the decryption key corresponding to an encryption key. By applying a decryption key to the key schedule, the round keys would be generated in the reverse order of that made by the encryption key. It is also necessary to note that the round keys generated for decryption are permuted versions of the round keys used in encryption, and this feature is required to provide the correct decryption round keys for the decryption process mentioned in the Section 8.2.3.

8.3 Security Analysis

In this section, we analyze the security strength of PUFFIN2 under differential and linear cryptanalysis and two major key schedule attacks.

8.3.1 Differential and Linear Cryptanalysis

Our proposed block cipher PUFFIN2 shares the same S-box and permutation mapping in the encryption and decryption process as PUFFIN, so the differential and linear cryptanalysis results of PUFFIN2 can be easily derived from that of PUFFIN in [60].

For differential cryptanalysis [61], the maximum differential characteristic probability of the S-box is $p_\delta = 1/4$ and, based on the 4×4 S-boxes and the involutional permutation, each round has at least one active S-box to form the path for a differential characteristic. Hence, the upper bound of the differential characteristic probability over 32 rounds is given by:

$$p_\Omega \leq p_\delta^{32} = 2^{-64}. \tag{8.6}$$

The differential characteristic probability p_Ω indicates that about 2^{64} chosen plaintext/ciphertext pairs would be required to mount a successful attack, the complexity of which is close to a brute force dictionary attack on a 64-bit cipher. Therefore, it is reasonable to consider PUFFIN2 to be resistant to differential cryptanalysis.

For linear cryptanalysis [62], the maximum linear approximation probability bias of the S-box is $|\varepsilon_S| = 1/4$. Similar to the case in differential cryptanalysis, there is at least one active S-box involved in each round to form a linear approximation. Hence, the upper bound of the linear approximation bias ε_L of 32 rounds is calculated with the piling-up lemma [62] as follows:

$$\varepsilon_L \leq 2^{31}|\varepsilon_S|^{32} = 2^{-33}. \tag{8.7}$$

According to [62], the number of the known plaintexts required to perform linear cryptanalysis is proportional to $1/\varepsilon_L^2$, which means an effective attack on PUFFIN2 with linear cryptanalysis requires about 2^{66} known plaintext/ciphertext pairs and therefore is considered to be an impractical attack.

It is necessary to mention that the success chance of the cryptanalysis may be underestimated by the probabilities of the differential and linear approximations calculated above. For differential cryptanalysis, the differential characteristic probability is calculated based on the assumption that the data entering different S-boxes are independent and there is only a single differential approximation path involved for the selected input and output differential patterns. In realistically, there are usually multiple differential approximation paths involved on the same input and output differential patterns so that the actual probability of the differential approximation is larger than the probability calculated when considering only a single approximation path. This concept is referred to as differentials [63]. Similarly, for linear cryptanalysis, the actual linear approximation probability is larger than the estimation above by taking into account the dependence of the S-box approximations and multiple linear approximation paths with the same plaintext bits and data bits at the input to the last round (referred to as a linear hull [64]).

8.3.2 Related-Key Attacks

The related-key attack was proposed in [65]. It exploits the regularity of the relationships between key schedule rounds and uses the chosen key relations to retrieve the secret key information. The related-key attack generally finds its application on those block ciphers that use the same algorithm to generate round keys for all the rounds, such as the variants of variants of LOKI [66] and Lucifer [67].

It is easy to see that our block cipher does not have this regularity property in the key schedule because the permutation layers PL64 and PR64 are not regularly distributed among rounds, and hence it is resistant to the related-key attack.

8.3.3 Weak Keys

Weak keys [68] are keys that make the key schedule produce identical round keys for all or some of the rounds. For the key schedule of PUFFIN2, due to the existence of nonlinear substitution layers, we do not find any weak key in the key space.

8.3.4 Updated Cryptanalysis Results

After PUFFIN and PUFFIN2 were proposed, the security strength of them has been analyzed by other researchers, as is shown in [69] and [70]. In [69], a linear cryptanalysis against PUFFIN is performed by taking into account linear hulls and recovers 4 bits of the last round key with the complexity less than 2^{58}. In [70], differential cryptanalysis is mounted on both PUFFIN and PUFFIN2. The cryptanalysis in [70] recovers the 80-bit key of PUFFIN2 with $2^{74.78}$ operations using $2^{52.3}$ chosen plaintexts. This cryptanalysis complexity is lower than the estimation in Section 8.3.1 since it takes into account multiple differentials (following the framework presented in [71]). The updated cryptanalysis results show the underestimation of the cost for key recover by our own analysis. As a simple way to enhance the security strength of PUFFIN2 in order to thwart the lower attack cost found in [70], the number of rounds of PUFFIN2 can be increased and in doing so the area of the hardware implementation of PUFFIN2 based on the loop-iterative structure is not increased.

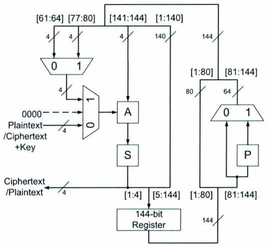


Figure 8.3: Serialized architecture of PUFFIN2

8.4 Serialized Architecture for Hardware Implementation

The proposed block cipher PUFFIN2 is designed to be efficiently implemented with a serialized hardware architecture. A serialized architecture is the architecture where multiple identical hardware components that work for multiple tasks simultaneously are mapped to one piece of the hardware component that works for the multiple tasks in series. Generally, a serialized architecture can lead to the minimal hardware implementation among a variety of implementation architectures. In this section, we introduce a serialized architecture for which the proposed block cipher PUFFIN2 is well suited in and that results in an ultra compact implementation.

The serialized architecture for PUFFIN2 is shown in Figure 8.3. This architecture is constructed with two 4-bit 2-to-1 multiplexers, a 64-bit 2-to-1 multiplexer, a 4-bit XOR adder A, a 4×4 S-box S, a 64-bit permutation P, and a 144-bit register. The architecture conceives a 144-bit wide datapath (for both data and key) and the hardware components operate on the datapath.

It is also worthy to mention that there is a 4-bit rotation structure on the datapath, which is crucial to ensure the hardware resources are shared properly in series and the block cipher algorithm is running correctly in the architecture. The 4-bit rotation structure is realized by crossover wiring that maps bits 1 to 140 to bits 5 to 144 and bits 141 to 144 to bits 1 to 4 (through the adder A and the S-box S).

The 4-bit 2-to-1 multiplexer with a dashed 4-bit zero input in Figure 8.3 is called the first 4-bit multiplexer and the other one is called the second 4-bit multiplexer. The first 4-bit multiplexer is able to output a zero vector independent of its two inputs, and this is achieved by ANDing the 4-bit output of the multiplexer with a signal bit from the controller.

In the next section, we describe the work flow of the serialized architecture with the example of plaintext and key loading procedure and the first round of the encryption process. We call the bits that carry plaintext information and key information during the encryption process as internal plaintext bits and internal key bits, respectively. A 64-bit plaintext plus an 80-bit encryption key is loaded in units of 4 bits through the first 4-bit multiplexer to the datapath. The first 4-bit unit is presented to the input of the 144-bit register in the first clock cycle and becomes available at the output of the register in the second clock cycle. Each subsequent 4-bit unit is added with a 4-bit zero vector and then fed through the S-box before being stored

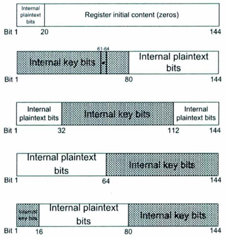


Figure 8.4: Contents of the 144-bit register at clock cycles 6, 37, 45, 53 and 57

in the 144-bit register. The 4-bit zero vector is generated by the initial output of the 144-bit register. The 4-bit rotation structure makes sure each 4-bit unit is added with a 4-bit zero vector and stored in the register bits next to the last 4-bit unit. The loading procedure of the plaintext plus the key takes 36 clock cycles and during this period the first substitution layers in the encryption process and key schedule are also performed. In the 37th clock cycle, the 64-bit internal plaintext bits are permuted with the 64-bit permutation by selecting position 1 of the 64-bit 2-to-1 multiplexer, and then the rightmost 4 bits of the updated internal plaintext bits are added with the rightmost 4 bits of the left 64 bits of the internal key bits by selecting position 0 of the second 4-bit multiplexer. It takes 16 clock cycles to complete the key addition of 64 bits and this ends at the 52nd clock cycle. In the 53rd cycle, the rightmost 4 bits of the 80-bit internal key bits are added with a 4-bit zero vector in A and then

Table 8.5: Implementation results of PUFFIN2 and serialized PRESENT

	Area (GEs)	Max. Freq. (MHz)	Clock Cycles per 64-bit block	Throughput @100 KHz
PUFFIN2	1083	326.8 MHz	1240	5.2 Kbps
Serialized PRESENT	1296	346.0 MHz	563	11.4 Kbps

substituted by the S-box. In the 57th cycle, the left 64 bits of the 80-bit internal key bits are permuted with the 64-bit permutation. In order to better demonstrate the work of the architecture, the contents of the 144-bit register at clock cycles 6, 37, 45, 53 and 57 are shown in Figure 8.4 (a), (b), (c), (d) and (e), respectively. The dotted 4-bit block in Figure 8.4 (b) is the 4 bits to be added with the rightmost 4 bits of the internal plaintext (after the permutation) in the 37th cycle.

The 64-bit internal plaintext bits and the 80-bit internal key bits are rotated within the 144-bit register and it takes 36 cycles to complete a full rotation. The period of 36 cycles is also the time to complete a round of the encryption/decryption process. Hence, the total time to complete the entire encryption/decryption including the 16 cycles for the initial loading of the plaintext is given by:

$$(16 + 36 \times 34)cc = 1240cc, \tag{8.8}$$

where cc represents one clock cycle.

8.5 Hardware Implementation Results

The block cipher PUFFIN2 with the serialized architecture has been implemented and synthesized with the 0.18-μm CMOS standard cell library from TSMC. Synopsys Design Compiler version X-2005.09 has been used as our synthesis tool. We also implemented the serialized PRESENT from [47] which is claimed as the smallest implementation of a block cipher with 64-bit block size. Both of the implementations are datapath-only implementations, which means their controllers are not included in the implementations, and in both cases the controllers are negligible because they can be realized with a small counter and a small amount of combinational logic. Our implementation results of PUFFIN2 and the serialized PRESENT are shown in Table 8.5. In the table, the metric of gate equivalents (GEs) is used, where a unit of 1 GE represents an area equivalent to a 2-input NAND gate.

According to Table 8.5, the implementation of PUFFIN2 is 16% smaller than the serialized PRESENT implementation. As a trade-off PUFFIN2 takes almost double the time of the serialized PRESENT to process the same amount of data. This is because the datapath of PUFFIN2 is reused for the key schedule and the datapath operation can not be performed simultaneously as the key schedule operation. However, in most lightweight applications, a large running time is not a serious issue.

It is necessary to point out that the gate count of the serialized PRESENT implementation claimed in [47] is 1075 GE. The 221 GE overhead of our implementation of the serialized PRESENT could be caused by the different synthesis library and the use of scan flip flops with integrated multiplexers in [47] instead of the normal flip flops and separated multiplexers found in our implementation. The same area reduc-

Table 8.6: Count of hardware components of PUFFIN2 and serialized PRESENT

Components	PUFFIN2	Serialized PRESENT
64-bit register (384 GE)	1 (35.5%)	1 (29.6%)
80-bit register (480 GE)	1 (44.3%)	1 (37.0%)
64-bit 2to1 multiplexer (153 GE)	1 (14.1%)	1 (11.8%)
80-bit 2to1 multiplexer (192 GE)	0	1 (14.8%)
4-bit 2to1 multiplexer (10 GE)	2 (1.8%)	3 (2.3%)
4x4 S-box (30 GE/32 GE)	1 (2.8%)	1 (2.5%)
4-bit XOR adder (11 GE)	1 (1.0%)	1 (0.9%)
5-bit XOR adder (14 GE)	0	1 (1.1%)
4 2-input AND gates (5 GE)	1 (0.5%)	0
Total gate count	1083 GE (100%)	1296 GE (100%)

tion effect can be achieved in our implementation of PUFFIN2 with scan flip flops as long as the 144-bit register is moved to the output of the 64-bit 2-to-1 multiplexer to form the integrated flip flops and multiplexers. The position of the 144-bit register is flexible in the serialized architecture, so this change would not have any influence on the functionality.

In order to have a clear comparison between the hardware complexity of PUFFIN2 and the serialized PRESENT, we list the count of the hardware components required for both implementations in Table 8.6. The 144-bit register in PUFFIN2 is divided into a 64-bit register and an 80-bit register in Table 8.6, and the 36 4-bit 2-to-1 multiplexers in the two shift registers of the serialized PRESENT are merged and shown as a 64-bit 2-to-1 multiplexer and an 80-bit 2-to-1 multiplexer in Table 8.6.

From Table 8.6, we can see the major area difference between PUFFIN2 and the serialized PRESENT comes from the 80-bit 2-to-1 multiplexer, which accounts for 14.8% of the total area of the serialized PRESENT and does not exist in PUFFIN2. It is also noticeable in Table 8.6 that the 144-bit register takes 80% of the hardware resource of PUFFIN2, and this fact allows us to believe that the serialized implementation of PUFFIN2 has approached the area limit of the block ciphers that have similar block size and key size.

8.6 Summary

In this chapter we have proposed a new block cipher PUFFIN2 based on an involutional SPN structure. The cipher with a 64-bit block size and an 80-bit key size can provide sufficient security for low cost embedded devices and support both encryption and decryption. We also introduced a serialized architecture based on which PUFFIN2 can be implemented with an ultra compact size. Compared with the serialized PRESENT implementation, the datapath of PUFFIN2 uses 16% fewer gates. In general, the PUFFIN2 block cipher is a secure, area-efficient structure in comparison to other proposed compact block ciphers.

Chapter 9

Conclusions

Implementation of AES with minimal resource cost or desired cost trade-off is typically the goal of hardware design of AES. This dissertation presents the investigation that helps to reach the goal from the aspect of implementation architecture. In the next section, this dissertation is concluded with the summary of research and contributions and the suggestions for future work.

9.1 Summary of Research and Contributions

The research and contributions of this dissertation include the following:

Performance Characterization of AES S-Box and Datapath Implementation Architectures

The research is the first work in literature that examines and compares the performance of an extensive range of AES S-box and datapath implementations in terms of timing, area, power and energy based on the same implementation technology. Previous research usually focuses on the techniques that improves certain perspectives of the performance based on a specific implementation architecture, rather than exploring the performance variation by different implementation architectures. The architecture range investigated in this dissertation covers most of the possible typical architectures of hardware AES implementation, which include the S-box pipeline configurations of component level 2 to 4 stages and gate level 2 to 7 stages and the datapath architectures with the widths of 8, 16, 32 and 64 bits and unrolling factors of 1, 2, 5 and 10 for 128-bit width. The performance of these implementations is characterized based on the same implementation platform under a variety of throughput constraints which covers a wide range of design requirements for various applications of AES. The performance characterization allows for better understanding of the influence on the performance from the aspect of implementation architecture. The characterization results show the extensive performance trade-offs offered by different implementation architectures and can serve as a general reference for flexible and efficient AES implementations.

Identification of Resource Efficient AES S-Box and Datapath Implementation Architectures

The research is the first work in literature that applies pipelining to the S-box implementation for resource-efficient purpose, instead of the high throughput purpose that is investigated in previous research. Obvious resource reduction in terms of area, power and/or energy is achieved with the appropriate pipeline configurations compared with the non-pipelined S-box implementation under the same throughput constraint. The research also identifies the power/energy efficient datapath architecture. It finds that the most energy efficient architecture is not achieved by the most low power architecture, which is used for both low power and low energy purposes previously. By combining the appropriate pipeline configuration and datapath architecture, the energy consumption is further significantly reduced compared with the most low power architecture.

Development of Novel and Efficient AES Datapath Architectures

The research develops the AES datapath architecture with 16, 32 and 64-bit width. While the 16 and 64-bit width architectures are newly presented, the 32-bit architecture has novel aspects which give the benefit of reduced storage and/or fewer clock cycles compared to previous work. All of these architectures are designed to complete the operation with the minimal number of clock cycles and with the minimal number or close to the minimal number of registers. No specific memory macro is required for these architectures since all the components are composed of standard cells. The development of these architectures contributes to the flexible implementation of AES.

Design of a Lightweight Block Cipher PUFFIN2

PUFFIN2 is a lightweight block cipher developed based on its predecessor PUFFIN. The cryptanalysis of PUFFIN2 shows that it can provide modest security strength suitable for many lightweight security applications. PUFFIN2 has a involutional structure and the datapath can be used for key generation. These features allow for very compact serialized hardware implementation. Compared with the previous most compact block cipher PRESENT, which requires different hardware for encryption and decryption operations, PUFFIN2 can be built with smaller area and has the same hardware for both encryption and decryption operations.

9.2 Suggestions for Future Work

Due to the limit of time, some further in-depth research has not been conducted and included in this dissertation. These work is mentioned in the following:

Investigation of Pipeline Configurations for Specific AES S-Box Structures

The investigation of pipeline configurations for the AES S-box is based on a typical composite field structure from [5] in order to show the typical effect on performance by pipelining. There exists a number of other composite field S-box structures, including [6], [7], [8], [9], [10] and [13], and these structures are usually elaborately developed to achieve certain performance benefits over the typical structure we adopt in this dissertation. It can be expected that investigating the pipeline configurations for each of the specific S-box structures would result in pipelined S-box implementations with better performance than those examined in this dissertation. It is also worthwhile to investigate new composite field structures that are specifically developed for pipelined structure. Moreover, the placement of pipeline registers examined in this dissertation is determined either by the synthesis tool at the gate level or by the virtual component boundary at the component level. There actually exists many more options for register placement and this can be explored and optimized for performance improvement. It is also necessary to conduct the performance comparison between the pipelined AES S-boxes based on composite field structures with the S-boxes of structures other than composite field structures, such as the decoder-permutation-encoder structure for low power purpose in [14]. In this way, a more complete picture of performance characteristics of the hardware implementation of the AES S-box is achieved.

Investigation of AES Datapath Architecture with Inner Round Pipeline

In this dissertation, we only investigate the architectures with outer round pipelining, i.e., pipeline that is only applied between two consecutive round functions for the 128-bit width datapath, and some cases of inner round pipelining (the pipeline within the round functions) by using the pipelined S-boxes (as is shown in Chapter 7). By considering the entire hardware implementation of the round function of the AES including *ShiftRows* and *MixColumns*, there is more datapath architecture pipeline options available for the exploration of better performance and more performance trade-offs.

Investigation of the Influence on Side Channel Attacks under Various AES Datapath Architectures

Due to the increasing significance of the ability to resist side channel attacks for block cipher implementations, it would be interesting to investigate the behaviour of the different AES datapath architectures under side channel attacks. Attention could be given to the investigation of the behaviour of the architectures under power analysis since these architectures have quite different power characteristics as are shown in this dissertation.

Design of the Successor of PUFFIN2 with Improved Security

According to the recently proposed cryptanalysis against PUFFIN2 in [70], PUFFIN2 does not fully achieve the originally claimed security strength and the vulnerability lies in the involutional property of the cipher. It would be desirable to design the new lightweight block cipher based on PUFFIN and PUFFIN2 that has the enhanced security resistent to the recently proposed cryptanalysis while still featuring an involutional structure so that the compactness and the support for both encryption and decryption remains.

Bibliography

[1] US National Insitute of Standards and Technology, "Advanced Encryption Standard," *Federal Information Processing Standards Publication, no. 197*, 2001.

[2] A. Bogdanov, D. Khovratovich, and C. Rechberger, "Biclique cryptanalysis of the full AES," in *Proceedings of the 17th international conference on The Theory and Application of Cryptology and Information Security*, Lecture Notes in Computer Science, pp. 344–371, Springer-Verlag, 2011.

[3] W. Stallings, *Cryptography and Network Security*. Prentice Hall, fourth ed., 2005.

[4] V. Rijmen, "Efficient Implementation of the Rijndael S-Box." http://www.esat.kuleuven.ac.be/rijmen/rijndael/.

[5] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC Implementation of the AES SBoxes," in *Proceedings of Topics in Cryptology (CT-RSA 2002)*, vol. 2271, pp. 67–78, 2002.

[6] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," in *Proceedings of Advances in Cryptology (ASIACRYPT 2001)*, vol. 2248 of *Lecture Notes in Computer Science*, pp. 239–254, 2001.

[7] D. Canright, "A Very Compact S-Box for AES," in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005)*, vol. 3659 of *Lecture Notes in Computer Science*, pp. 441–455, 2005.

[8] X. Zhang and K. K. Parhi, "On the Optimum Constructions of Composite Field for the AES Algorithm," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, pp. 1153–1157, 2006.

[9] S. Nikova and V. Rijmen, and M. Schlaffer, "Using Normal Bases for Compact Hardware Implementations of the AES S-Box," in *Proceedings of the 6th Conference on Security and Cryptography for Networks (SCN 2008)*, pp. 236–245, 2008.

[10] M. M. Kermani and A. Reyhani-Masoleh, "A Low-Cost S-box for the Advanced Encryption Standard Using Normal Basis," in *Proceedings of the IEEE International Conference on Electro/Information Technology (EIT 2009)*, pp. 52–55, 2009.

[11] M. M. Wong, M. L. D. Wong, A. K. Nandi, and I. Hijazin, "Construction of Optimum Composite Field Architecture for Compact High-Throughput AES S-Boxes," *IEEE Transactions on Very Large Scale Intergration (VLSI) Systems*, vol. 99, pp. 1–5, 2011.

[12] M. M. Wong, M. L. D. Wong, A. K. Nandi, and I. Hijazin, "Composite Field $GF(((2^2)^2)^2)$ Advanced Encryption Standard (AES) S-box with Algebraic Normal Form Representation in the Subfield Inversion," *IET Circuits Devices System*, vol. 5, pp. 471–476, 2011.

[13] Y. Nogami, K. Nekado, T. Toyota, N. Hongo, and Y. Morikawa, "Mixed Bases for Efficienct Inversion in $GF((2^2)^2)^2$ and Conversion Matrices of SubBytes of AES," in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2010)*, Lecture Notes in Computer Science, pp. 234–247, 2010.

[14] G. Bertoni, M. Macchetti, L. Negri, and P. Fragneto, "Power-Efficient ASIC Synthesis of Cryptographic S-Boxes," in *Proceedings of the 14th ACM Great Lakes symposium on VLSI (GLSVLSI04)*, pp. 277–281, 2004.

[15] S. Morioka and A. Satoh, "An Optimized S-box Circuit Architecture for Low Power AES Design," in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002)*, Lecture Notes in Computer Science, pp. 271–295, Springer-Verlag, 2003.

[16] Y. Zeng, X. Zou, Z. Liu, and J. Lei, "A Low-Power Rijndael S-box Based on Pass Transmission Gate and Composite Field Arithmetic," *Journal of Zhejiang University - Science A*, vol. 8, pp. 1553–1559, 2007.

[17] S. Morioka and A. Satoh, "A High Throughput Low Power Compact AES S-Box Implementation Using Composite Field Arithmetic and Algebraic Normal Form Representation," in *Proceedings of the 2nd Asia Symposium on Quality Electronic Design (ASQED 2010)*, pp. 318–323, 2010.

[18] S. Tillich, M. Feldhofer, T. Popp, and J. Großschädl, "Area, Delay, and Power Characteristics of Standard-cell Implementations of the AES S-Box," *Journal of Signal Processing Systems*, vol. 50, pp. 251–261, 2008.

[19] X. Zhang and K. Parhi, "High-speed VLSI Architectures for the AES Algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, pp. 957–967, 2004.

[20] A. Hodjat and I. Verbauwhede, "Area-Throughput Trade-offs for Fully Pipelined 30 to 70 Gbits/s AES Processors," *IEEE Transactions on Computers*, vol. 55, pp. 366–372, 2006.

[21] T. Good and M. Benaissa, "AES on FPGA from the Fastest to the Smallest," in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005)*, vol. 3659 of *Lecture Notes in Computer Science*, pp. 427–440, 2005.

[22] T. Good and M. Benaissa, "Pipelined AES on FPGA with Support for Feedback Modes (in a Multi-Channel Environment)," *IET Information Security*, vol. 1, pp. 1–10, 2007.

[23] K. U. Järvinen, M. T. Tommiska, and J. O. Skyttä, "A Fully Pipelined Memoryless 17.8 Gbps AES-128 Encryptor," in *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays*, pp. 207–215, 2003.

[24] N. Iyer, P. Anandmohan, D. Poornaiah, and V. Kulkarni, "High Throughput, Low Cost, Fully Pipelined Architecture for AES Crypto Chip," in *Proceedings of the 2006 Annual IEEE India Conference*, pp. 1–6, 2006.

[25] M.-Y. Wang, C.-P. Su, C.-L. Horng, C.-W. Wu, and C.-T. Huang, "Single- and Multi-core Configurable AES Architectures for Flexible Security," *IEEE Trans-*

actions on Very Large Scale Integration (VLSI) Systems, vol. 18, pp. 541–552, 2010.

[26] S. Mangard, M. Aigner, and S. Dominikus, "A Highly Regular and Scalable AES Hardware Architecture," *IEEE Transactions on Computers*, vol. 52, pp. 483–491, 2003.

[27] P. Chodowiec and K. Gaj, "Very Compact FPGA Implementation of the AES Algorithm," in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003)*, vol. 2779 of *Lecture Notes in Computer Science*, pp. 319–333, 2003.

[28] N. Pramstaller and J. Wolkerstorfer, "A Universal and Efficient AES Co-processor for Field Programmable Logic Arrays," in *Proceedings of the 14th Annual International Conference on Field-Programmable Logic and Applications (FPL 2004)*, pp. 565–574, 2004.

[29] C. Chang, C. Huang, K. Chang, Y. Chen, and C. Hsieh, "High Throughput 32-bit AES Implementation in FPGA," in *Proceedings of the 9th IEEE Asia Pacific Conference on Circuits and Systems (APCCAS 2008)*, pp. 1806–1809, 2008.

[30] J. R. V. Feldhofer and M. Wolkerstorfer, "AES Implementation on a Grain of Sand," *IET Information Security*, vol. 152, pp. 13–20, 2005.

[31] P. Hämäläinen, T. Alho, M. Hännikäinen, and T. D. Hämäläinen, "Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core," in *Proceedings of the 9th EUROMICRO Conference on Digital System Design (DSD 2006)*, pp. 577–583, 2006.

[32] T. Good and M. Benaissa, "692-nW Advanced Encryption Standard (AES) on a 0.13-μm CMOS," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, pp. 1753–1757, 2010.

[33] D. A. Group, *Digital ASIC Design: A Tutorial on the Design Flow.* Lund University, Sweden, 2005.

[34] Synopsys, *Design Compiler User Guide Version D-2010.03-SP2.* 2010.

[35] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Low Power Methodology Manual: For System-on-Chip Design.* Springer, 2007.

[36] Synopsys, *PrimeTime PX User Guide Version D-2010.06.* 2010.

[37] F.-X. Standaert, G. Piret, G. Rouvroy, J.-J. Quisquater, and J.-D. Legat, "ICEBERG : An Involutional Cipher Efficient for Block Encryption in Reconfigurable Hardware," in *Proceedings of the 11th International Workshop on Fast Software Encryption (FSE 2004)*, Lecture Notes in Computer Science, pp. 279–299, Springer-Verlag, 2004.

[38] C. Wang and H. Heys, "Using a Pipelined S-Box in Compact AES Hardware Implementations," in *Proceedings of the 8th IEEE International NEWCAS Conference*, pp. 101–104, 2010.

[39] A. Hodjat, D. D. Hwang, B. Lai, K. Tiri, and I. Verbauwhede, "A 3.84 Gbits/s AES Crypto Coprocessor with Modes of Operation in a 0.18-μm CMOS Technology," in *Proceedings of the 15th ACM Great Lakes symposium on VLSI (GLSVLSI 2005)*, pp. 60–63, 2005.

[40] S. K. Mathew, F. Sheikh, M. Kounavis, S. Gueron, A. Agarwal, S. K. Hsu, H. Kaul, M. A. Anders, and R. K. Krishnamurthy, "53 Gbps Native Composite-Field AES-Encrypt/Decrypt Accelerator for Content-Protection in 45 nm High-Performance Microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 46, pp. 767–776, 2011.

[41] A. van der Werf, J. L. van Meerbergen, E. H. L. Aarts, W. F. J. Verhaegh, and P. E. R. Lippens, "Efficient Timing Constraint Derivation for Optimal Retiming High Speed Processing Units," in *Proceedings of the 7th international symposium on High-level synthesis*, pp. 48–53, 1994.

[42] J. Leijten, J. van Meerbergen, and J. Jess, "Analysis and Reduction of Glitches in Synchronous Networks," in *Proceedings of the 1995 European conference on Design and Test*, pp. 398–401, 1995.

[43] J. Zambreno, D. Nguyen, and A. Choudhary, "Exploring Area/Delay Tradeoffs in an AES FPGA Implementation," in *Proceedings of the 14th Annual International Conference on Field-Programmable Logic and Applications (FPL 2004)*, pp. 575–585, 2004.

[44] P. Hämäläinen, T. Alho, M. Hännikäinen, T. D. Hämäläinen, T. Jävinen, P. Salmela, and J. Takala, "Efficient Byte Permutation Realizations for Compact AES Implementations," in *Proceedings of the 13th European Signal Processing Conference (EUSIPCO 2005)*, 2005.

[45] H. Li and Z. Friggstad, "An Efficient Architecture for the AES Mix Columns Operation," in *Proceedings of the 2005 IEEE International Symposium on Circuits and Systems (ISCAS 2005)*, pp. 4637–4640, 2005.

[46] C. Wang and H. M. Heys, "An Ultra Compact Block Cipher For Serialized Architecture Implementations," in *Proceedings of the 2009 Canadian Conference on Electrical and Computer Engineering (CCECE 2009)*, pp. 1086–1090, 2009.

[47] A. Bogdanov, L. Knudsen, G. Leander, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher," in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007)*, vol. 4272 of *Lecture Notes in Computer Science*, pp. 450–466, 2007.

[48] C. Rolfes, A. Poschmann, G. Leander, and C. Paar, "Ultra-Lightweight Implementations for Smart Devices - Security for 1000 Gate Equivalents," in *Proceedings of Smart Card Research and Advanced Application Conference (CARDIS 2008)*, vol. 5189 of *Lecture Notes in Computer Science*, pp. 89–103, 2008.

[49] C. Loon and T. Korkishko, "mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors," in *Proceedings of Information Security Applications (WISA 2005)*, vol. 3786 of *Lecture Notes in Computer Science*, pp. 243–258, 2008.

[50] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee, "HIGHT: A New Block Cipher Suitable for Low Resource Device," in *Proceedings of the International Workshop*

on Cryptographic Hardware and Embedded Devices (CHES 2006), vol. 4249 of *Lecture Notes in Computer Science*, pp. 46–59, 2006.

[51] F.-X. Standaert, G. Piret, N. Gershenfeld, and J.-J. Quisquater, "SEA: A Scalable Encryption Algorithm for Small Embedded Applications," in *Proceedings of Smart Card Research and Applications (CARDIS 2006)*, vol. 3298 of *Lecture Notes in Computer Science*, pp. 222–236, 2006.

[52] H. Cheng, H. Heys, and C. Wang, "PUFFIN: A Novel Compact Block Cipher Targeted to Embedded Digital Systems," in *Proceedings of the 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD 2008)*, pp. 383–390, 2008.

[53] C. Canniere, O. Dunkelman, and M. Knezevic, "KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers," in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2009)*, Lecture Notes in Computer Science, pp. 272–288, Springer-Verlag, 2009.

[54] M. Izadi, B. Sadeghiyan, S. Sadeghian, and H. Khanooki, " MIBS: A New Lightweight Block Cipher," in *Proceedings of Cryptology and Network Security 2009 (CANS 2009)*, Lecture Notes in Computer Science, pp. 334–348, Springer-Verlag, 2009.

[55] W. Wu and L. Zhang, "LBlock: A Lightweight Block Cipher," in *Proceedings of the 9th International Conference on Applied Cryptography and Network Security*

(ACNS 2011), Lecture Notes in Computer Science, pp. 327–344, Springer-Verlag, 2011.

[56] C. Shannon, "Communication Theory of Secrecy Systems ," *Bell System Technical Journal*, vol. 28, pp. 656–715, 1949.

[57] D. R. Stinson, *Cryptography Theory and Practice.* CRC Press, third ed., 2006.

[58] H. Feistel, "Cryptography and Computer Privacy," *Scientific American*, vol. 228, 1973.

[59] J. B. Kam and G. I. Davida, "Structured Design of Substitution-Permutation Encryption Networks," *IEEE Transactions on Computers*, vol. C-28, 1979.

[60] H. Cheng, *Compact Hardware Implementation of Block Cipher with Concurrent Error Detection.* Master Thesis, Facult of Engineering and Applied Science, Memorial University, 2007.

[61] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," in *Proceedings of Advances in Cryptology (CRYPTO 1990)*, vol. 537 of *Lecture Notes in Computer Science*, pp. 2–21, 1991.

[62] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," in *Proceedings ofAdvances in Cryptology (EUROCRYPT 1993)*, vol. 765 of *Lecture Notes in Computer Science*, pp. 386–397, 1994.

[63] X. Lai, J. L. Massey, and S. Murphy, "Markov Ciphers and Differential Cryptanalysis," in *Proceedings of Advances in Cryptology (EUROCRYPT 1991)*, Lecture Notes in Computer Science, pp. 17–38, Springer-Verlag, 1991.

[64] K. Nyberg, "Linear Approximations of Block Ciphers," in *Proceedings of Advances in Cryptology (EUROCRYPT 1994)*, Lecture Notes in Computer Science, pp. 439–444, Springer-Verlag, 1995.

[65] E. Biham, "New Type of Cryptanalysis Attacks Using Related Keys," in *Proceedings of Advances in Cryptology (EUROCRYPT 1993)*, vol. 765 of *Lecture Notes in Computer Science*, pp. 229–246, 1994.

[66] L. Brown, J. Pieprzyk, and J. Seberry, "LOKI: A Cryptographic Primitive for Authentication and Secrecy Applications," in *Proceedings of Advances in Cryptology (AUSCRYPT 1990)*, Lecture Notes in Computer Science, pp. 229–236, 1990.

[67] A. Sorkin, "Lucifer: a Cryptographic Algorithm," *IEEE Transactions on Computers*, vol. 8, pp. 22–41, 1984.

[68] J. H. Moore and G. J. Simmons, "Cycle Structure of the DES for Keys Having Palindromic (or Antipalindromic) Sequences of Round Keys," *IEEE Transactions on Software Engineering*, vol. SE-13, pp. 262–273, 1987.

[69] G. Leander, "On Linear Hulls, Statistical Saturation Attacks, PRESENT and a Cryptanalysis of PUFFIN," in *Proceedings of Advances in Cryptology (EUROCRYPT 2011)*, Lecture Notes in Computer Science, pp. 303–322, Springer-Verlag, 2011.

[70] C. Blondeau and B. Gérard, "Differential Cryptanalysis of PUFFIN and PUFFIN2," in *ECRYPT Workshop on Lightweight Cryptography 2011*, pp. 35–54, 2011.

[71] C. Blondeau and B. Gérard, "Multiple Differential Cryptanalysis: Theory and Practice," in *Proceedings of the International Workshop on Fast Software Encryption (FSE 2011)*, vol. 6733 of *Lecture Notes in Computer Science*, pp. 35–54, 2011.

Appendix A

Description of the Operation of the *ShiftRows* Components

The operation of the *ShiftRows* components shown in Figures 6.2, 6.3, 6.4 and 6.5 is controlled through the multiplexers. All the 8-bit registers are driven with a continuous clock. In order to demonstrate the operation of these components, the contents of the registers at some selected clock cycles are shown in Tables A.1, A.2, A.3 and A.4 for Figures 6.2, 6.3, 6.4 and 6.5, respectively, where the first clock cycle is denoted as CC00 and the n-th clock cycle after CC00 is denoted as CCn. The content of a register is a byte of the *State* following the notation in Figure 2.1.

Table A.1: Contents of the registers of the 8-bit width *ShiftRows* component at the selected clock cycles

	R01	R02	R03	R04	R05	R06	R07	R08	R09	R10	R11	R12
CC00	$B_{0,0}$	$B_{1,0}$	$B_{2,0}$	$B_{3,0}$	$B_{0,1}$	$B_{1,1}$	$B_{2,1}$	$B_{3,1}$	$B_{0,2}$	$B_{1,2}$	$B_{2,2}$	$B_{3,2}$
CC01	$B_{1,0}$	$B_{2,0}$	$B_{3,0}$	$B_{0,1}$	$B_{1,1}$	$B_{2,1}$	$B_{3,1}$	$B_{0,2}$	$B_{1,2}$	$B_{2,2}$	$B_{3,2}$	$B_{0,3}$
CC04	$B_{0,1}$	$B_{1,0}$	$B_{2,0}$	$B_{3,0}$	$B_{0,2}$	$B_{1,2}$	$B_{2,1}$	$B_{3,1}$	$B_{0,3}$	$B_{1,3}$	$B_{2,3}$	$B_{3,2}$
CC05	$B_{1,0}$	$B_{2,0}$	$B_{3,0}$	$B_{0,2}$	$B_{1,2}$	$B_{2,1}$	$B_{3,1}$	$B_{0,3}$	$B_{1,3}$	$B_{2,3}$	$B_{3,2}$	$B'_{0,0}$
CC09	$B_{1,0}$	$B_{2,0}$	$B_{3,1}$	$B_{0,3}$	$B_{1,3}$	$B_{2,1}$	$B_{3,2}$	$B'_{0,0}$	$B'_{1,0}$	$B'_{2,0}$	$B'_{3,0}$	$B'_{0,1}$
CC12	$B_{0,3}$	$B_{1,0}$	$B_{2,1}$	$B_{3,2}$	$B'_{0,0}$	$B'_{1,0}$	$B'_{2,0}$	$B'_{3,0}$	$B'_{0,1}$	$B'_{1,1}$	$B'_{2,1}$	$B'_{3,1}$
CC16	$B'_{0,0}$	$B'_{1,0}$	$B'_{2,0}$	$B'_{3,0}$	$B'_{0,1}$	$B'_{1,1}$	$B'_{2,1}$	$B'_{3,1}$	$B'_{0,2}$	$B'_{1,2}$	$B'_{2,2}$	$B'_{3,2}$

Table A.2: Contents of the registers of the 16-bit width *ShiftRows* component at the selected clock cycles

	R01	R02	R03	R04	R05	R06	R07	R08	R09	R10	R11	R12
CC00	$B_{0,0}$	$B_{1,0}$	$B_{2,0}$	$B_{3,0}$	$B_{0,1}$	$B_{1,1}$	$B_{2,1}$	$B_{3,1}$	$B_{0,2}$	$B_{1,2}$	$B_{2,2}$	$B_{3,2}$
CC01	$B_{2,0}$	$B_{3,0}$	$B_{0,1}$	$B_{1,0}$	$B_{2,1}$	$B_{3,1}$	$B_{0,2}$	$B_{1,2}$	$B_{2,2}$	$B_{3,2}$	$B_{0,3}$	$B_{1,3}$
CC02	$B_{0,1}$	$B_{1,0}$	$B_{2,1}$	$B_{3,0}$	$B_{0,2}$	$B_{1,2}$	$B_{2,0}$	$B_{3,1}$	$B_{0,3}$	$B_{1,3}$	$B_{2,3}$	$B_{3,2}$
CC03	$B_{2,1}$	$B_{3,0}$	$B_{0,2}$	$B_{1,0}$	$B_{2,0}$	$B_{3,1}$	$B_{0,3}$	$B_{1,3}$	$B_{2,3}$	$B_{3,2}$	$B'_{0,0}$	$B'_{1,0}$
CC05	$B_{2,0}$	$B_{3,1}$	$B_{0,3}$	$B_{1,0}$	$B_{2,1}$	$B_{3,2}$	$B'_{0,0}$	$B'_{1,0}$	$B'_{2,0}$	$B'_{3,0}$	$B'_{0,1}$	$B'_{1,1}$
CC06	$B_{0,3}$	$B_{1,0}$	$B_{2,1}$	$B_{3,2}$	$B'_{0,0}$	$B'_{1,0}$	$B'_{2,0}$	$B'_{3,0}$	$B'_{0,1}$	$B'_{1,1}$	$B'_{2,1}$	$B'_{3,1}$
CC08	$B'_{0,0}$	$B'_{1,0}$	$B'_{2,0}$	$B'_{3,0}$	$B'_{0,1}$	$B'_{1,1}$	$B'_{2,1}$	$B'_{3,1}$	$B'_{0,2}$	$B'_{1,2}$	$B'_{2,2}$	$B'_{3,2}$

Table A.3: Contents of the registers of the 32-bit width *ShiftRows* component at the selected clock cycles

	R01	R02	R03	R04	R05	R06	R07	R08	R09	R10	R11	R12
CC00	$B_{0,0}$	$B_{1,0}$	$B_{2,0}$	$B_{3,0}$	$B_{0,1}$	$B_{1,1}$	$B_{2,1}$	$B_{3,1}$	$B_{0,2}$	$B_{1,2}$	$B_{2,2}$	$B_{3,2}$
CC01	$B_{0,1}$	$B_{1,0}$	$B_{2,1}$	$B_{3,0}$	$B_{0,2}$	$B_{1,2}$	$B_{2,0}$	$B_{3,1}$	$B_{0,3}$	$B_{1,3}$	$B_{2,3}$	$B_{3,2}$
CC02	$B_{0,2}$	$B_{1,0}$	$B_{2,0}$	$B_{3,1}$	$B_{0,3}$	$B_{1,3}$	$B_{2,1}$	$B_{3,2}$	$B'_{0,0}$	$B'_{1,0}$	$B'_{2,0}$	$B'_{3,0}$
CC03	$B_{0,3}$	$B_{1,0}$	$B_{2,1}$	$B_{3,2}$	$B'_{0,0}$	$B'_{1,0}$	$B'_{2,0}$	$B'_{3,0}$	$B'_{0,1}$	$B'_{1,1}$	$B'_{2,1}$	$B'_{3,1}$
CC04	$B'_{0,0}$	$B'_{1,0}$	$B'_{2,0}$	$B'_{3,0}$	$B'_{0,1}$	$B'_{1,1}$	$B'_{2,1}$	$B'_{3,1}$	$B'_{0,2}$	$B'_{1,2}$	$B'_{2,2}$	$B'_{3,2}$

Table A.4: Contents of the registers of the 64-bit width *ShiftRows* component at the selected clock cycles

	R01	R02	R03	R04	R05	R06	R07	R08
CC00	$B_{0,0}$	$B_{1,0}$	$B_{2,0}$	$B_{3,0}$	$B_{0,1}$	$B_{1,1}$	$B_{2,1}$	$B_{3,1}$
CC01	$B_{0,2}$	$B_{1,0}$	$B_{2,0}$	$B_{3,2}$	$B_{0,3}$	$B_{1,3}$	$B_{2,1}$	$B_{3,1}$
CC02	$B'_{0,0}$	$B'_{1,0}$	$B'_{2,0}$	$B'_{3,0}$	$B'_{0,1}$	$B'_{1,1}$	$B'_{2,1}$	$B'_{3,1}$

Appendix B

Description of the Operation of the *MixColumns* Components

The operation of the *MixColumns* components shown in Figures 6.6, 6.7 and 6.8 is controlled through the multiplexers and the AND gates. All the 8-bit registers are driven with a continuous clock. In order to demonstrate the operation of these components, the contents of the registers at the clock cycles of an operation are shown in Tables B.1, B.2 and B.3 for Figures 6.6, 6.7 and 6.8, respectively, where the first clock cycle is denoted as CC00 and the n-th clock cycle after CC00 is denoted as CCn. The content of a register is a byte following the notation in (2.2).

Table B.1: Contents of the registers of the 8-bit width *MixColumns* component for the clock cycles during a complete operation ($m = n + 1$)

	R01	R02	R03	R04	R05	R06	R07
CC00	$B_{0,n} \oplus 02B_{1,n} \oplus 03B_{2,n} \oplus B_{3,n}$	$B_{0,n} \oplus B_{1,n} \oplus 02B_{2,n} \oplus 03B_{3,n}$	$03B_{0,n} \oplus B_{1,n} \oplus B_{2,n} \oplus B_{2,n}$	$B_{0,m}$	$B_{0,m}$	$03B_{0,m}$	$02B_{0,m}$
CC01	$B_{0,n} \oplus B_{1,n} \oplus 02B_{2,n} \oplus 03B_{3,n}$	$03B_{0,n} \oplus B_{1,n} \oplus B_{2,n} \oplus B_{2,n}$	don't care	$B_{0,m} \oplus B_{1,m}$	$03B_{0,m} \oplus B_{1,m}$	$02B_{0,m} \oplus 03B_{1,m}$	$B_{0,m} \oplus 02B_{1,m}$
CC02	$03B_{0,n} \oplus B_{1,n} \oplus B_{2,n} \oplus B_{2,n}$	don't care	don't care	$03B_{0,m} \oplus B_{1,m} \oplus B_{2,m}$	$02B_{0,m} \oplus 03B_{1,m} \oplus B_{2,m}$	$B_{0,m} \oplus 02B_{1,m} \oplus 03B_{2,m}$	$B_{0,m} \oplus B_{1,m} \oplus 02B_{2,m}$
CC03	don't care	don't care	don't care	$02B_{0,m} \oplus 03B_{1,m} \oplus B_{2,m} \oplus B_{3,m}$	$B_{0,m} \oplus 02B_{1,m} \oplus 03B_{2,m} \oplus B_{3,m}$	$B_{0,m} \oplus B_{1,m} \oplus 02B_{2,m} \oplus 03B_{3,m}$	$03B_{0,m} \oplus B_{1,m} \oplus B_{2,m} \oplus B_{2,m}$

Table B.2: Contents of the registers of the 16-bit width *MixColumns* component for the clock cycles during a complete operation ($m = n + 1$)

	R01	R02	R03
CC00	$B_{0,n} \oplus B_{1,n} \oplus$ $02B_{2,n} \oplus 03B_{3,n}$	$B_{0,m} \oplus B_{1,m}$	$02B_{0,m} \oplus 03B_{1,m}$
CC01	don't care	$02B_{0,m} \oplus 03B_{1,m}$ $\oplus B_{2,m} \oplus B_{3,m}$	$B_{0,m} \oplus B_{1,m} \oplus$ $02B_{2,m} \oplus 03B_{3,m}$

	R04	R05	R06
CC00	$03B_{0,n} \oplus B_{1,n}$ $\oplus B_{2,n} \oplus B_{2,n}$	$03B_{0,m} \oplus B_{1,m}$	$B_{0,m} \oplus 02B_{1,m}$
CC01	don't care	$B_{0,m} \oplus 02B_{1,m} \oplus$ $03B_{2,m} \oplus B_{3,m}$	$03B_{0,m} \oplus B_{1,m}$ $\oplus B_{2,m} \oplus B_{2,m}$

Table B.3: Contents of the registers of the 32-bit width *MixColumns* component for the clock cycles during a complete operation

	R01	R02	R03	R04
CC00	$02B_{0,n} \oplus 03B_{1,n}$ $\oplus B_{2,n} \oplus B_{3,n}$	$B_{0,n} \oplus 02B_{1,n} \oplus$ $03B_{2,n} \oplus B_{3,n}$	$B_{0,n} \oplus B_{1,n} \oplus$ $02B_{2,n} \oplus 03B_{3,n}$	$03B_{0,n} \oplus B_{1,n}$ $\oplus B_{2,n} \oplus B_{2,n}$