Looting the LUTs : FPGA Optimization of AES and AES-like Ciphers for Authenticated Encryption

Mustafa Khairallah[1], Anupam Chattopadhyay[1,2], and Thomas Peyrin[1,2]

[1] School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore

[2] School of Computer Science and Engineering
Nanyang Technological University, Singapore
mustafam001@e.ntu.edu.sg,anupam@ntu.edu.sg,thomas.peyrin@ntu.edu.sg

Abstract. In this paper, we investigate the efficiency of FPGA implementations of AES and AES-like ciphers, specially in the context of authenticated encryption. We consider the encryption/decryption and the authentication/verification structures of OCB-like modes (like OTR or SCT modes). Their main advantage is that they are fully parallelisable. While this feature has already been used to increase the throughput/performance of hardware implementations, it is usually overlooked while comparing different ciphers. We show how to use it with zero area overhead, leading to a very significant efficiency gain. Additionally, we show that using FPGA technology mapping instead of logic optimization, the area of both the linear and non linear parts of the round function of several AES-like primitives can be reduced, without affecting the run-time performance. We provide the implementation results of two multi-stream implementations of both the LED and AES block ciphers. The AES implementation in this paper achieves an efficiency of 38 Mbps/slice, which is the most efficient implementation in literature, to the best of our knowledge. For LED, achieves 2.5 Mbps/slice on Spartan 3 FPGA, which is 2.57x better than the previous implementation. Besides, we use our new techniques to optimize the FPGA implementation of the CAESAR candidate Deoxys-I in both the encryption only and encryption/decryption settings. Finally, we show that the efficiency gains of the proposed techniques extend to other technologies, such as ASIC, as well.

Keywords: AES, FPGA, Authenticated Encryption, Logic Optimization, Technology Mapping, Deoxys, LED

1 Introduction

In September 2016, the CAESAR competition committee announced the selection of 15 Authenticated Encryption with Associated Data (AEAD) schemes as candidates for round 3 of the CAESAR competition [1]. This competition signifies the current need for practical, secure and efficient AEAD schemes. An AEAD scheme typically consists of two routines. The first one is encryption $\mathcal{E}_K(AD, M)$ which takes as input a shared key K, public associated data AD and the message to be encrypted M and returns a tagged ciphertext C. The second one is decryption/verification $\mathcal{D}_K(AD, C)$, which either returns an invalid symbol $\perp$ if the received ciphertext, associated data and the authentication data do not match, or the decrypted message M, otherwise.

An important aspect of the study of AEAD schemes is the evaluation of their hardware performance, which clearly needs more efforts. So far, nearly all candidates have been supported with a basic hardware implementation [2]. However, the implementations are done on various platforms, for different interfaces and thus, a comprehensive evaluation is still missing. Furthermore, several designs have unique advantages to offer in some platforms, e.g., Field Programmable Gate Array (FPGA), which is not fully exploited. This is one of the prime motivations for this manuscript.

In the survey presented in [3], the authors classified the round 2 candidates of the CAESAR competition into five families according to their base constructions: block cipher-based, stream cipher-based, key-less permutations, hash-function-based and dedicated schemes. In this paper, we focus on the block cipher-based family. Specifically, we focus on optimizations for algorithms that allow block-level parallelism while using the underlying block cipher, such as the Offset Code Book mode (`OCB`) [4–6], the Synthetic Counter-in-Tweak mode (`SCT`) [7] and the Offset Two-Round mode (`OTR`) [8].

All the available hardware implementations of the CAESAR competition candidates on the ATHENa hardware evaluation website [2] are fully sequential implementations, i.e. to start processing a new block, all the previous blocks have to be finished. These implementations do not take full advantage of the specific characteristics of the schemes based on the aforementioned modes.

Generally, circuit optimization consists of two phases: logic synthesis and technology mapping. For certain target technologies, such as FPGA, logically optimized circuits do not provide the optimal mapping to the underlying technology, leaving behind a lot of under-utilized hardware resources. This phenomenon is obvious in the `AES` Sbox circuits proposed by Boyar [9, 10], which are logical optimizations of the circuit proposed by Canright [11]. These circuits are much smaller than the straight-forward ROM-based Sbox in terms of gate count and circuit depth. These two features make them the natural choice for low area Application-Specific Integrated Circuits (ASIC) implementations of `AES`. Interestingly, on the other hand, practical results show that one can achieve a smaller area on FPGA by using the ROM approach [12]. By analyzing this result, it appears that due to the specific details of these circuits [9–11], it is hard to map them efficiently to look-up tables (LUTs) that the FPGA is constructed from, leading to a lot of under-utilized/unusable logic gates inside the FPGA.

In Figure 1, the number of LUTs required for implementing two 8-bit to 8-bit ROM-based Sboxes (which are both the forward and inverse `AES` Sboxes)

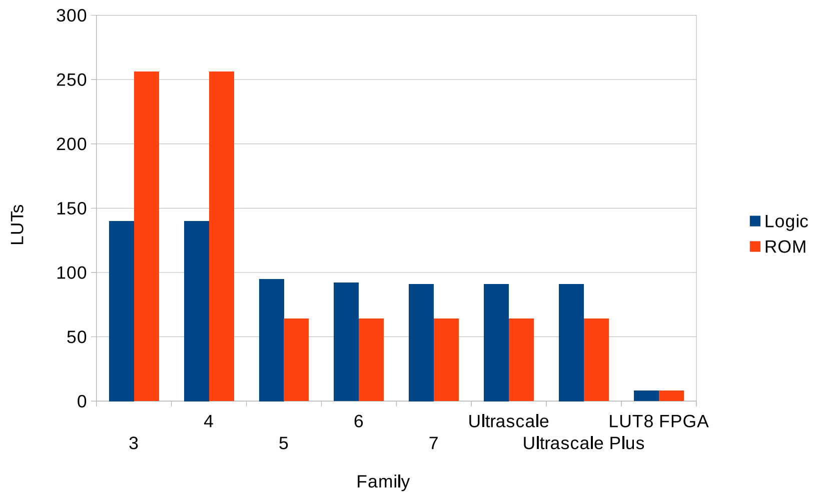

Fig. 1: Evolution of the `AES` Sbox/ISbox area vs. Xilinx FPGA families

is compared with the implementation of Boyar's shared encryption/decryption Sbox [9]. These results are plotted against the technology evolution of Xilinx FPGAs as an example. Analysing the chart, it is clear that after the introduction of the Virtex 5 family, logic optimization of the Sbox stopped being beneficial. The reason for that was the introduction of the 6-input LUTs, which enabled implementing an 8-to-1 look-up table using only four 6-input LUTs and three dedicated multiplexers, or five 6-input LUTs. In other words, the ROM-based Sbox has become both faster and smaller than the logic-based Sbox, even when both encryption and decryption are implemented using a shared data path. While the technology seems to be saturated around the 6-input LUT structure, a hypothetical family has been added to the chart assuming 8-input LUT structure, showing that such a family will make the cost of both logic-based and ROM-based implementations exactly the same (8 LUTs). While these results may seem specific to Xilinx FPGAs, other vendors, e.g. Altera, also use 6-input LUTs as their building blocks and will follow the same trend. Besides, the FPGA industry seems to be saturated around this building block and we believe that the same trend will follow for the upcoming years.

In a nutshell, the current implementations for multiple designs in the CAESAR contest do neither exploit the underlying block-level parallelism and nor consider the FPGA-specific optimizations. Both of these shortcomings are addressed in the current manuscript, achieving significant gain in area-efficiency, run-time performance or both.

Our Contributions. In this article, we propose new improvements for FPGA implementations of AEAD schemes based on AES-like primitive. These improvements are twofold.

Firstly, we provide a new efficient hardware architecture for OCB-like AEAD modes (Section 2). The architecture uses a generic multi-stream AES-like cipher, such as AES or Deoxys-BC (the tweakable block cipher used in CAESAR competition candidate Deoxys [13]) as an underlying primitive. This architecture can be easily modified to support the OTR or SCT AEAD modes for example.

Secondly, we improve the implementation efficiency of several AES-like ciphers, such as AES, LED and Deoxys-BC. In particular, the problem of FPGA mapping and under-utilized hardware discussed earlier is studied in details for two applications (Section 3):

- we show how to design low-area logic primitives optimized for FPGA LUTs instead of the number of logic gates (Section 3.2).
- we explain how to select the locations of pipelining registers to accommodate as many independent streams as possible without any additional area cost compared to the single stream architecture (Section 3.3).

Eventually, as practical results, following these implementation strategies we obtained very efficient LED and AES implementations (Section 4). For example, our AES implementation achieves an efficiency of 38 Mbps/slice, which is the most efficient AES FPGA implementation in the literature to the best of our knowledge. We also applied our techniques to Deoxys, and we obtained the current best Deoxys-I FPGA implementation, improving their efficiency by a factor ~1.7 with almost the same area. Table 1 shows a summary of our results compared to state of the art implementations.

2 OCB Multi-stream Architecture

2.1 OCB Mode Description

Notation. m_i is the i^{th} plaintext message block. c_i is the i^{th} ciphertext block. AD_i is the i^{th} AD block. T_i is the tweak value related to the i^th block of the message or AD. K is the shared secret key. $\mathcal{E}_{K,T_i}$ is the bock cipher used by the AEAD algorithm. $\sum m_i$ is the XOR checksum of the message.

This section includes a simplified description of the OCB mode. An interested reader may refer to [4] for a full description. The OCB AEAD mode consists of two parts, shown in Figures 2 and 3. In the original proposal [4–6], first the associated data is processed using the PMAC [19] structure shown in Figure 3. Second, the message is encrypted using the structure in Figure 2, computing the message checksum in parallel. Finally, the message checksum in encrypted and XOR-ed to the associated data tag to produce the final tag. These two structures, with minor changes, appear also in other encryption modes, such as OTR, SCT, CTR etc. Therefore, the ideas and techniques presented in this paper can be also to beneficial for these other modes. Before describing the architecture, we present observations that inspired the architecture:

Table 1: Summary of our results compared to the state-of-the-art implementations

Algorithm	Family	Impl.	Throughput (Gbps)	Slices	Efficiency (Mbps/slice)
AES Encryption	Virtex 5	Section 4.1	8.0	347	23.00
		[14]	4.5	400	11.20
		[15]	46.0	3,579	12.88
	Virtex 6	Section 4.1	9.5	247	38.46
		[15]	64.1	3.121	20.55
AES Decryption	Virtex 5	Section 4.1	6.1	294	20.7
		[14]	4.5	550	7.6
Deoxys-I-128	Virtex 6	Section 4.2	3.8	861	4.5
		[16]	2.2	946	2.57
Deoxys-I-128 Encryption Only	Virtex 6	Section 4.2	3.5	566	6.2
		[17]	1	920	1.12
LED	Spartan 3	Section 4.3	0.51	204	2.5
		[18]	0.19	204	0.97

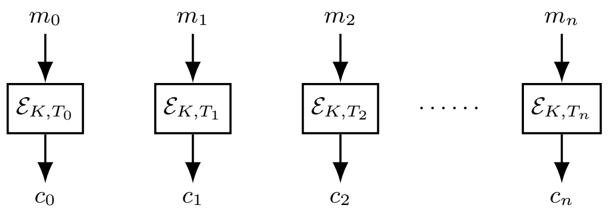

Fig. 2: The parallel encryption structure

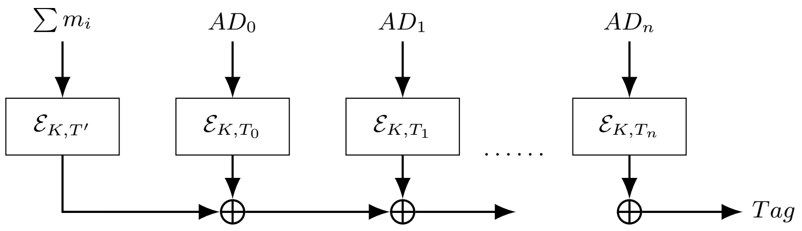

Fig. 3: The encrypt-then-xor construction

1. The first and second parts of execution do not depend on each other. Consequently, following the implementation from Poschmann and Stöttinger on

the ATHENa website [2], the order can be reversed. This enables one to use the same storage for both the checksum and tag computation.

2. In Figure 2 the computations are completely independent, while in Figure 3, there is an output dependency between different blocks. Since there is no input dependency, both the structures are fully parallelisable. Additionally, a small temporal shift saves the temporary storage needed. For example, the first block starts at time $t = 0$ and the second block starts at $t = \Delta t$. At $t = T$ the first block is finished and stored in the tag storage. Finally, at time $t = T + \Delta t$ the second block is finished and XOR-ed with tag, in-place.

2.2 Related Work

The Cryptographic Engineering Research Group (CERG) at George Mason University (GMU), USA, runs and maintains the online platform ATHENa [16] aimed at fair, comprehensive, and automated evaluation of hardware cryptographic cores targeting FPGAs, All Programmable Systems on Chip, and ASICs. One of their on-going projects is the comparison of FPGA implementations of the CAESAR competition candidates. They have also provided high-speed round-based implementations of round 2 candidates. Among these candidates, several use `OCB`-like modes: `OCB` v1.1[20] and `AES-OTR` v3.1 [21] (which use `AES` as underlying cipher), and `Deoxys-I` v1.41[13] (which uses an `AES`-like tweakable block cipher `Deoxys-BC`). `Deoxys-BC` uses the same data path as `AES` but defines a new tweak/key-schedule that requires a smaller number of gates to evaluate when compared to `AES` (but with an additional 128-bit tweak input). It also requires a higher number of rounds compared to `AES`.

The implementations provided by the CERG team are round-based implementations that compute one cipher round per cycle. These implementations are compliant with the CAESAR Hardware API [22], developed for fair comparison among CAESAR candidates. On the other hand, a round-based implementation of `Deoxys-I` (encryption only) was provided by Poschmann and Stöttinger, that is not compliant with the required API. One of the requirements of the CAESAR Hardware API is to load the encryption/decryption key into the hardware core at most once per message. Since the implementation from Poschmann and Stöttinger [17] does not follow the API, it permits loading the key again with every message block, allowing the designers to get rid of the master key storage, saving 128 flip-flops. They also save 128 extra flip-flops by noticing that during the tag computation, the encryption of checksum can be computed before the associated data. This enables using the same storage for the message checksum and the intermediate tag value, saving 128 more flip flops. We follow the later approach in our implementation due to its obvious area advantage. We will see later in Section 4.2 that even though we target the CAESAR Hardware API compliance, our implementation of the `Deoxys-I-128` encryption-only algorithm has better results compared to both the previous implementations.

2.3 Proposed Architecture

The proposed high level architecture is shown in Figure 4. For simplicity, only the encryption data path is drawn. However, a similar data path for decryption can also be included. The architecture consists of a single round of the underlying block cipher, which is divided into N stages, each stage takes one cycle to be processed. If the block cipher requires r rounds, the architecture loads and processes N blocks, every $r \cdot N$ rounds, which leads an average latency of r cycles, equivalent to a simple single round implementation. The selection of N depends on several considerations:

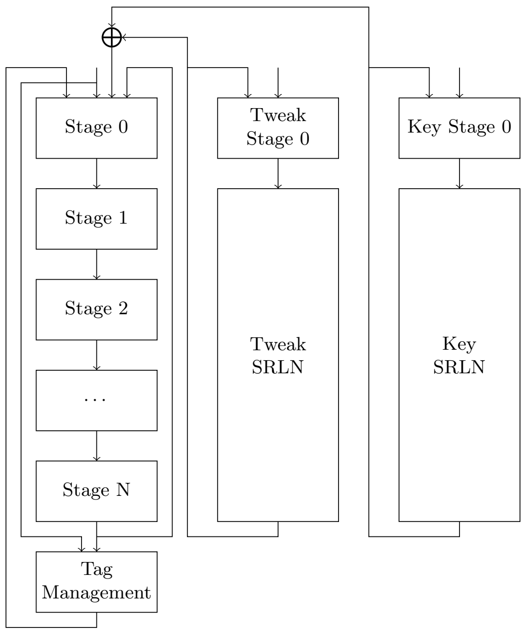

Fig. 4: Multi-stream OCB Hardware Architecture

1. This architecture is intended for high speed over long messages. It is noticeable that any number of blocks less than N requires the time to be encrypted. Consequently, a very large N leads to a huge overhead for short messages or for messages whose block length is not divisible by N.
2. In order to minimize the key scheduling overhead, it is performed in only one pipeline stage and then shifted N cycles. This is based on the SRL feature of the FPGA LUTs, which allows the utilization of very compact serial shift registers using logic LUTs. For most FPGAs, a single LUT can implement either a 16-bit or 32-bit SRL, which we consider as the upper bound on the value of N.

3. The pipeline registers can add a huge overhead over the simple round implementation. Therefore, in Section 3.3 we describe a technique to select the optimal locations of the pipeline registers in the FPGA implementation.

From these three considerations, we concluded that the optimal value for N is between 2 and 4, neglecting the control overhead. This leads to a speed-up between 2x and 4x. Additionally, for applications that require ultra high speed over very long messages, e.g. disk encryption, high speed multimedia interfaces, etc., and do not care about the area, the same architecture can be unrolled into a fully pipelined implementation. This can lead to a huge increase of the throughput. Specifically, the single round multi-stream architecture requires about $r \cdot N \cdot \lceil \frac{B}{N} \rceil$ cycles to compute B blocks. On the other hand, a fully unrolled architecture has an initial latency of $r \cdot N$ and a new block is generated every cycle, leading to a total number of cycles of $r \cdot N + B - 1$. The speed up over the round implementation is given by

$$G = \frac{r \cdot B}{r \cdot N + B - 1}$$

and for very long messages, the unrolled architecture has a speed up of r times. Since the area increases less than r times (only the round part is replicated while the tag and control part almost have the same area), the efficiency remains unchanged. In Section 4.1 we show that an `AES` round can be implemented with a clock frequency greater than 700 MHz on FPGA, with almost the same number of slices/LUTs. Therefore, we estimate that this variant can be suitable for applications that require very high speed authenticated encryption.

3 Multi-Stream `AES`-like Ciphers

3.1 AES Data Path State-of-the-Art FPGA Implementations

`AES` [23] is a 128-bit block cipher, standardized in 2001 by NIST. It is based on the Substitution-Permutation Network (SPN). The internal state of the cipher can be viewed as a 4×4 matrix of bytes. It consists of 10 SPN rounds. Each round includes a `SubBytes` operation for the non-linear part, `ShiftRows` and `MixColumns` for the linear permutation and `AddRoundKey` for key addition. `SubBytes` consists of 16 independent 8-bit Sboxes, `ShiftRows` shifts the bytes in each row, independently, and `MixColumns` applies a diffusion matrix to each column, independently. All byte operations are done in GF(2^8).

In this section, we quickly review state-of-the-art high speed `AES`-128 FPGA implementations (we only discuss full width round-based and unrolled implementations). A detailed survey on `AES` data paths for FPGA is provided in [12]. Full-width FPGA implementations of `AES` are either unrolled implementations [15], round-based single stream [24, 25] or round-based multi-stream [14]. Although the scope of this paper is round-based multi-stream implementations, the optimizations described in this section can be used for any of the aforementioned implementations. In [14], the authors proposed the `AES` data path shown in Figure 5. Each box in Figure 5 represents a pipeline stage, and it can be noticed that the selection of the pipeline stages is based on the functionality of each stage, which leads to two very fast stages in the beginning, then two slow stages afterwards. This limits the maximum possible frequency. In the next sections, we will show why this architecture might not be optimal and describe a new four-stream data path designed for FPGA to achieve higher performance efficiency.

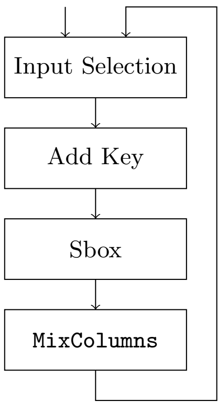

Fig. 5: The `AES` encryption data path from [14]

3.2 LUT-based Optimization of Linear Transformations

Notation. a, b, c and d are the four bytes that compose one column of the `AES` state. a_i is the i^{th} bit of a, where a_0 is the least significant bit. Upper-case letters A, B, C, D, E, F are the hexadecimal representations of the decimal values 10, 11, 12, 13, 14 and 15, respectively. $\cdot$ and $\oplus$ are multiplication and addition over GF(2^8).

The `AES` `MixColumns` circuit is a matrix multiplication operation of the `AES` state byte matrix by a constant matrix M given by

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

which is a circulant MDS matrix. For `AES` 128-bit architectures, the `MixColumns` operation can be viewed as 16 dot-products of the vector $\begin{bmatrix} 2\,3\,1\,1 \end{bmatrix}$ and a vector composed by a permutation of 4 state words. It can also be viewed as four 32-bit to 32-bit mappings (four matrix-vector products over state vectors). The later view is favorable for ASIC implementations, as it allows reducing the required number of gates by sharing many intermediate results of the computation.

Specifically, only 108 XOR gates are required for implementing the 32-bit mapping [26]. However, as discussed earlier, since modern FPGAs use big 6/5 input LUTs to implement logic circuits, having a lot of small shared 2/3-input gates is not the most efficient solution. Synthesizing the circuit used in [26] or [27] for Virtex-6 FPGA requires 41 LUTs for low area and 44 LUTs for high speed. On the other hand, the dot-product view is given by

$$p = 2 \cdot a \oplus 3 \cdot b \oplus c \oplus d$$

which can be decomposed into

$$\begin{bmatrix} a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & 0 \\ 0 & 0 & 0 & a_7 & a_7 & 0 & a_7 & a_7 \\ b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 & 0 \\ 0 & 0 & 0 & b_7 & b_7 & 0 & b_7 & b_7 \\ b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 \\ d_7 & d_6 & d_5 & d_4 & d_3 & d_2 & d_1 & d_0 \end{bmatrix} \tag{1}$$

where the elements of each column represent the inputs of one output function. From this perspective, it can be seen that 5 outputs can be implemented using one 5-input LUT, while 3 outputs can be implemented using 7-input LUT, which can be implemented using two 6-input LUTs. That sums to a total of 11 LUTs per output coefficient, 44 LUTs per output column. This shows that logic optimization does not offer much gain over the straightforward implementation of the transformation. Besides, a deeper look at the view given by the decomposition in (1) shows that the three outputs that need 7-input LUTs share two inputs bits, namely $a_7 \& b_7$. Decomposition (1) can be written as decomposition (2), where $x = a_7 \oplus b_7$. This decomposition can be implemented using eight 6-input LUTs and one 2-input LUT, a total of 9 LUTs per output coefficient, 36 LUTs per output column (which is smaller than the best-reported implementations) or 1.125 LUTs per output bit. It is worth mentioning that this number is near-optimal for any linear transformation over 32 bits, as the optimal number is 1 LUT/bit, which corresponds to transformation where each output bit depends on n bits, where $2 \leq n \leq 6$ (the case where $n = 1$ corresponds to an identity function and can be neglected, w.l.o.g.)[3].

[3] In fact, each 6:1 LUT can be implemented as a 5:2 LUT with shared inputs. Using this feature, our circuit can be indeed implemented using only 8 LUTs, which is the optimal figure. However, in this paper we are handling the optimization at the front-end stage and this feature is incorporated automatically by the placement and routing tool.

$$\begin{bmatrix} a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & a_7 \\ b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 & b_7 \\ 0 & 0 & 0 & x & x & 0 & x & 0 \\ b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 \\ d_7 & d_6 & d_5 & d_4 & d_3 & d_2 & d_1 & d_0 \end{bmatrix} \tag{2}$$

The optimization of the `AES` inverse `MixColumns` circuit is less straightforward, as M^{-1} includes larger coefficients. M^{-1} is given by

$$\begin{bmatrix} E & B & D & 9 \\ 9 & E & B & D \\ D & 9 & E & B \\ B & D & 9 & E \end{bmatrix}$$

A lot of work has been done on how to reuse the same circuit from M to implement M^{-1} with minimal overhead. This is done by using any of the following relations $M^{-1} = M^3$, $M^{-1} = M \cdot N$ or $M^{-1} = M \oplus K$, where N and K are matrices with low coefficients. In that direction, the circuit given by decomposition (2) will also be the smallest and the same reasoning can be used to achieve small area for both K and N. However, this approach is most useful for low area serial implementations with shared encryption/decryption data path. They do not achieve the best results for high speed round implementations with dedicated decryption data path. For example, using $M^{-1} = M^3$ requires 3.375 LUTs/bit and produces a large-depth circuit (low performance), while using $M^{-1} = M \oplus K$ is even larger. The most promising approach is $M^{-1} = M \cdot N$ which requires 288 LUTs/block, corresponding to 2.25 LUTs/bit, which is still far from optimal. On the other hand, the straightforward implementation of M^{-1} leads to output functions that include 19 input bits, which can lead to very low performance. Here, we give a circuit that requires 60 LUTs per output column, corresponding to 1.875 LUTs/bit. First, we use the same dot product view mentioned earlier, which is given by equation (3).

$$p' = E \cdot a \oplus B \cdot b \oplus D \cdot c \oplus 9 \cdot d = F \cdot (a \oplus b \oplus c \oplus d) \oplus (a \oplus 4 \cdot b \oplus 2 \cdot c \oplus 4 \cdot d \oplus 2 \cdot d) \tag{3}$$

Second, two observations are made

1. $F \cdot (a \oplus b \oplus c \oplus d)$ is constant across any output column.
2. $4 \cdot (a \oplus c)$ is shared by two output coefficients. The same is valid for $4 \cdot (b \oplus d)$.

Using these two observations, a circuit that requires only 60 LUTs per output column can be implemented. The circuit diagram is given in Figure 6. This is 17% smaller than the best reported implementation. Given that `MixColumns` is the main difference between the `AES` encryption and decryption data paths, optimizing this primitive is crucial. On the other hand, since 1.875 LUTs/bit is still far from the optimal 1 LUT/bit figure, there may be some room for further optimization.

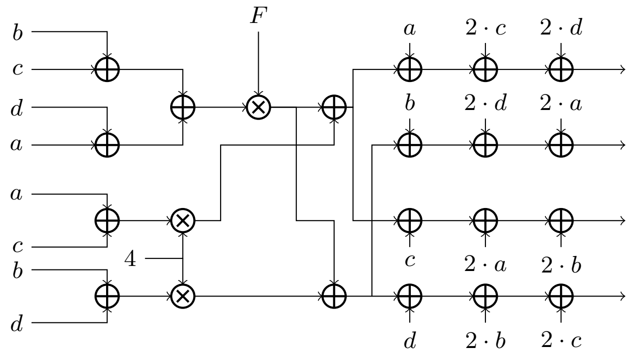

Fig. 6: FPGA-friendly inverse `MixColumns` circuit

3.3 Zero Area Overhead Pipelining

Pipelining has been used by hardware designers/architects as a tool to increase throughput/run-time performance for a long time. However, a fully pipelined block cipher implementations can be costly, due to the large area requirements. A more realistic approach is to use multi-stream implementations. These implementations start from a sequential implementation that processes one block in C cycles, and divides it into N pipeline stages. This leads to computing x blocks in $N \cdot C$ cycles, where $x \in \{1, 2, ..., N\}$. x depends on the number of independent block streams the user can leverage. However, this is a double-edged weapon, due to the following reasons:

1. The time required to process one block in a sequential implementation is $\sim C \cdot T$, where T is the critical path delay of the implementation. If the N pipeline stages divide the critical path evenly into segments of $\frac{T}{N}$ delay, the time required to process N blocks becomes $T + t$, where t is a small overhead, leading to $\sim N$x speed-up. Unfortunately, the critical path is usually not evenly divided, leading to a sub-optimal speed-up ($< N$).
2. Modern FPGA families consist of a basic building block called LUT6, which is a 6-input single-output look-up table. Additionally, each unit of this building block has an associated Flip-Flop, which the designer/synthesis tool can choose to either use it or not. In Figure 7, we show the optimal utilization of a LUT6 unit in a pipelined architecture, where it is used to implement a 6-input circuit followed by storing the output. On the other hand, in Figure 8, a poor selection of the location of pipeline stage is in-place, leading to the utilization of 3 look-up tables, instead of 1 in the case of Figure 7. In other words, the poor choice of where to place the pipeline registers leads to a significant increase in area.

While the impact of moving the flip-flops 1 logic level forward in the previous example is obvious, the designers usually do not have an accurate estimation of the exact LUT utilization before synthesis. Consequently, the designers choose

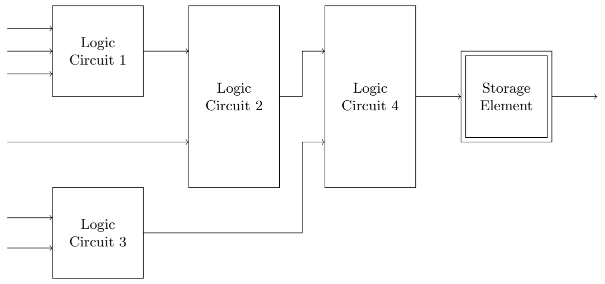

Fig. 7: Optimal pipeline selection

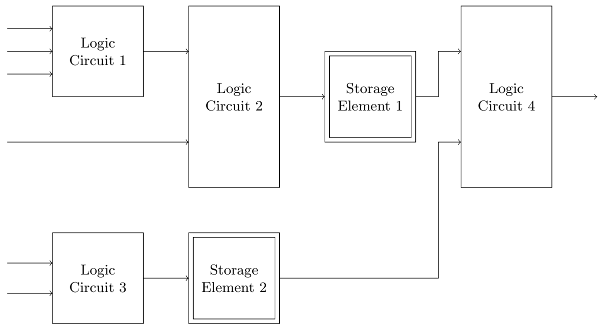

Fig. 8: Sub-optimal pipeline selection

the pipeline stages based on the logical functions, e.g. Sbox, `MixColumns`, input selection, etc. In our work, we follow a different approach. First, we synthesize a single stream sequential implementation of the required block cipher. Second, we study the output layout to to determine the precise distribution of pipeline stages without affecting the structure of the utilized LUTs.[4].

4 Implementations and Results

4.1 Two-Stream and Four-Stream `AES` Implementations

Using the techniques described in Sections 3.2 and 3.3, we have implemented two multi-stream `AES` data path (two and four streams), shown in Figure 9. In addition to the use of the low area `MixColumns` circuit and ROM-based Sbox, the locations of the pipelining registers have been selected specifically to ensure as efficient LUT utilization as possible. In other words, both the two-stream and four-stream implementations use the same number of logical LUTs (944 LUTs, without key scheduling), out of which 95% (896 LUTs) are 6-input LUTs. The

[4] The term zero overhead refers to the number of LUT-FF pairs, as this is the important metric, not the number of LUTs or FFs.

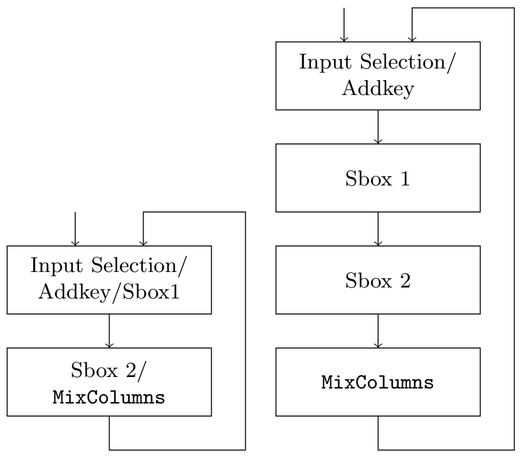

Fig. 9: Two/Four-Stream `AES` round data path implementation

results are given in Table 2. The comparison is restricted to full-width implementations that do not utilize any BRAMs. For that reason, the implementation in [28] is not included. While it has a very high efficiency (yet smaller than ours, $\sim$ 30 Mbps/slice), this number is biased due to the use of BRAMs to reduce the number of slices. It can be observed that our data path, with only two-streams, outperforms the data path from [14] (with four streams) in terms of both efficiency and area and reaches the same throughput. This result is achieved due to more than one factor:

1. The critical path in the implementation from [14] consists of three levels of logic inside the Sbox used (one LUT6 followed by MUX7 and MUX8). In our design, the critical path also consists of three levels of logic (Sbox part 2 (LUT6) and the `MixColumns` circuit proposed in Section 3.2 (LUT3 + LUT6)).
2. The `MixColumns` circuit used is smaller.
3. The first two pipeline stages in Figure 5 necessitate the use of 256 LUTs. The two stages altogether can be viewed as a 6-input function, which can be easily merged into a single stage of 128 LUTs (of type LUT6).

This implies that our proposed implementation achieves the same performance as [14] for lower latency and using only independent two streams (easier to achieve). In fact, following the architecture in Section 2, it can be used even for slightly dependent streams (even and odd blocks of an `OCB` message). Additionally, by choosing to add two more stages at the output of the Sbox 2 and key addition circuits, the performance and efficiency can be further enhanced without any additional increase in the area occupied, as shown in Table 2. The results shows that our four-stream implementation outperforms all the `AES` FPGA implementations in the literature in terms of efficiency, to the best of our knowledge. In Table 2, we show the implementation results of the `AES` decryption data path.

It is shown that it has a speed-up of around 2x over the similar implementation from [14].

Table 2: Implementation results of AES on Virtex-5/6 FPGA (encryption only)

Family	Implementation	Key schedule	Number of Slices	Max. freq (MHz)	Throughput (Gbps)	Efficiency (Mbps/slice)
Virtex 5	Ours/2 streams	Offline	347	350	4.5	12.90
	Ours/4 streams	Offline	347	625	8.0	23.00
	[14]/4 streams	Offline	400	350	4.5	11.20
	[15]/ unrolled	Offline	3579	360	46.0	12.88
Virtex 6	Ours/4 streams	Offline	347	752	9.6	27.66
	Ours/4 streams	No	247	752	9.5	38.46
	[15]/ unrolled	Offline	3121	501	64.1	20.55

Table 3: Implementation results of AES on Virtex-5 FPGA (decryption only)

Implementation	Key schedule	Number of Slices	Max. freq (MHz)	Throughput (Gbps)	Efficiency (Mbps/slice)
Ours/4 streams	No	294	477	6.1	20.7
Ours/4 streams	Offline	445	477	6.1	13.7
[14]/4 streams	Offline	550	350	4.5	7.6

4.2 Round-based Two-Block Deoxys-I-128

As mentioned earlier, the Deoxys-I CAESAR candidate uses an underlying tweakable block cipher called Deoxys-BC. This cipher is similar to AES, with three major differences:

1. It consists of 14 rounds instead of 10.
2. The final round includes a MixColumns operation, as opposed to AES.
3. It uses a different key schedule, which is smaller than the AES key schedule, but uses an extra public tweak value.

Based on the architecture proposed in Section 2 and the AES data paths proposed in Section 4.1, we have implemented two complete data paths for Deoxys-I. They include two and four pipeline stages, respectively. They also consist of four parts: the encryption data path, the decryption data path, the key schedule and the tweak schedule. Using the pipeline selection technique, both implementations consume the same area, except for the decryption data path which we implement only for 4 streams. In Table 4, it is shown that the bottleneck of the design, not considering the control overhead, is the decryption data path.

Table 4: Results of the `Deoxys-I-128` data path implementation on Virtex-6 FPGA

Block	Number of LUT-FF pairs	Max. Freq. (2-stream)	Max. Freq. (4-stream)
Enc. data path	1455	492 MHz	785 MHz
Dec. data path	1170	-	665 MHz
Key Schedule	442	724 MHz	724 MHz
Tweak Schedule	413	620 MHz	620 MHz

Based on this datapath, a full implementation of `Deoxys-I-128` has been implemented on Xilinx Virtex-6 FPGA (xc6vlx75tff784-3). The overall utilization is $\sim$ 3800 LUT-FF pairs and maximum operating frequency is 454 MHz. This 27% performance degradation from the datapath estimated clock frequency comes from two reasons:

1. While the datapath is very fast, there is still optimization required to the control unit to cope with such speed.
2. Although we choose a small FPGA from the Virtex 6 family, the design is still small compared to the FPGA size, which leads to it becoming I/O dominated. leading to a lot of wiring delays related to the I/O pins. This will not be applicable if the design is used as a part of a larger on-chip system.

To verify the second problem as part of the reason for performance degradation, we also implemented the design for the small Spartan-6 (xc6slx9ftg256-3) FPGA. The maximum operating frequency is 273 MHz vs. 333 MHz pre-layout (only 18% degradation). The results of the Virtex-6 implementation are summarized in Table 5. For fair comparison, we have also downloaded and implemented the `Deoxys-I-128` implementation reported on the ATHENa website [2] by CERG team. We only compare the cipher circuits without the overhead of the hardware API. Our results show an efficiency gain of 75% (1.75x) for Virtex 6 and 74% (1.74x) for Spartan-6. Table 5 shows the results for the encryption-only implementation. Our implementation is 5.536x more efficient than the implementation by (Poschmann and Stöttinger).

Table 5: Post-layout results of the `Deoxys-I-`128 implementation on FPGA

FPGA	Impl.	Number of Slices	Max. Freq. (MHz)	Throughput (Mbps)	Efficiency (Mbps/Slice)
Virtex 6	Ours	861	454	3,874	4.5
	[16]	946	285	2,432	2.57
Spartan 6	Ours	1,010	273	2,329	2.31
	[16]	1,032	161	1,373	1.33

Table 6: Post-layout results of the `Deoxys-I-`128 encryption only implementation on Virtex 6 FPGA

Impl.	Number of Slices	Max. Freq. (MHz)	Throughput (Mbps)	Efficiency (Mbps/Slice)
Ours	566	416	3,549	6.2
[17]	920	161	1.030	1.12

Relevance to the CAESAR Competition The goal of this section is not to show that Deoxys-I is faster than other CAESAR candidates. The goal is to show that parallelism can actually increase the efficiency of parallelisable ciphers by significant factors. Since we are comparing two different architectures and implementations of the same cipher, it makes more sense to focus the comparison on the cipher itself. While the API provides a fair methodology for comparing different ciphers, it can hide the potential of enhancement for a certain cipher. This is exactly true for FPGA in our case, as a custom control unit that is both compliant with the CAESAR API and the 4-stream architecture has to be designed. This leads to 50% decrease in the performance of the proposed architecture during the CAESAR Competition Round 3 benchmarking stage. We emphasize that this is not the goal of the proposed architecture, as in real-life applications faster APIs can be designed, such as the one we used in the previous comparison.

Extending the results to other technologies (ASIC) Since it can be argued that the techniques and optimizations in this paper are limited because they are limited to a certain technology (Xilinx FPGA), we have synthesized both the proposed implementation and the implementation from [16] for ASIC using Synopsys Design Compiler and the TSMC 65nm technology. The results show that even with full API compliance, the proposed implementation has 54% higher throughput and is 38% more efficient. These results are summarized in

Table 7. We are currently in the process of preparing the HDL code to be provided publicly soon so that other researchers can verify our results.

Table 7: Synthesis results of the `Deoxys-I`-128 implementation using TSMC 65nm technology

Impl.	Area (KGE)	Max. Freq. (MHz)	Throughput (Mbps)	Efficiency (Mbps/KGE)
Ours	59.53	847	7,227	121.40
[16]	53.37	549	4,684	87.76

4.3 Three-Stream `LED` Implementation

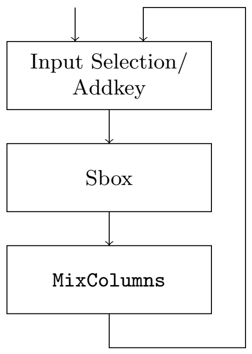

Fig. 10: Three-Stream (χ^4) `LED`-64 round implementation

LED [29] is a 64-bit block cipher based on an `AES`-like SPN. Its state is a 4×4 matrix of 4-bit nibbles. In this paper we focus on the 64-bit key version `LED`-64. However, the same results can extend to the other variants of `LED`, since the only difference is the key scheduling part, which can be easily adjusted for this architecture. The (χ^4) round implementation from [18], Section 3.1, has been replicated for Spartan-3 Xilinx FPGA. Using the guidelines from Section 3.3, we have been able to add two extra pipeline stages at the outputs of the Sbox and the `MixColumns` operations, as shown in Figure 10. In Table 8, it is shown that almost all the available flip-flops has been used, increasing both the throughput and efficiency by 2.57x at no additional area cost.

Table 8: Results of the three-stream LED-64 implementation compared to the single stream counterpart on Spartan 3 FPGA.

Implementation	Number of Slices	Number of FFs	Max. freq (MHz)	Throughput (Mbps)	Efficiency (Mbps/slice)
[18]/1 stream	204	74	98.7	197.35	0.97
Ours/3 streams	204	202	257	514	2.5

Acknowledgments

The authors would like to thank the anonymous referees for their helpful comments. This work is partly supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

1. CAESAR Competition: CAESAR submissions. https://competitions.cr.yp.to/caesar-submissions.html (2016)
2. George Mason University: ATHENa: Automated Tools for Hardware EvaluatioN. https://cryptography.gmu.edu/athena/ (2017)
3. Abed, F., Forler, C., Lucks, S.: General classification of the authenticated encryption schemes for the CAESAR competition. Computer Science Review (2016)
4. Rogaway, P., Bellare, M., Black, J.: OCB: A block-cipher mode of operation for efficient authenticated encryption. ACM Transactions on Information and System Security (TISSEC) **6**(3) (2003) 365–403
5. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: International Conference on the Theory and Application of Cryptology and Information Security, Springer (2004) 16–31
6. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: International Workshop on Fast Software Encryption, Springer (2011) 306–327
7. Peyrin, T., Seurin, Y.: Counter-in-tweak: authenticated encryption modes for tweakable block ciphers. In: Annual Cryptology Conference. (2016)
8. Minematsu, K.: Parallelizable rate-1 authenticated encryption from pseudorandom functions. In: EUROCRYPT, Springer (2014) 275–292
9. Boyar, J., Peralta, R.: A new combinational logic minimization technique with applications to cryptology. In: International Symposium on Experimental Algorithms, Springer (2010) 178–189
10. Boyar, J., Peralta, R.: A small depth-16 circuit for the AES s-box. In: IFIP International Information Security Conference, Springer (2012) 287–298
11. Canright, D.: A very compact s-box for AES. In: International Workshop on Cryptographic Hardware and Embedded Systems, Springer (2005) 441–455
12. Resende, J.C., Chaves, R.: AES Datapaths on FPGAs: A State of the Art Analysis. In: Hardware Security and Trust. Springer (2017) 1–25

13. Jean, J., Nikolic, I., Peyrin, T., Seurin, Y.: Deoxys v1.41. Technical report, Nanyang Technological University, Singapore/ANSSI, Paris, France (2016)
14. Bulens, P., Standaert, F.X., Quisquater, J.J., Pellegrin, P., Rouvroy, G.: Implementation of the AES-128 on Virtex-5 FPGAs. In: International Conference on Cryptology in Africa, Springer (2008) 16–26
15. Liu, Q., Xu, Z., Yuan, Y.: A 66.1 gbps single-pipeline aes on fpga. In: 2013 International Conference on Field-Programmable Technology (FPT). (Dec 2013) 378–381
16. : Deoxys-I-128 implementation by cerg team. https://cryptography.gmu.edu/athena/ (2016)
17. Poschmann, A., Stöttinger, M.: Deoxys-I-128 implementation by poschmann and Stöttinger. https://cryptography.gmu.edu/athena/ (2016)
18. Anandakumar, N.N., Peyrin, T., Poschmann, A.: A very compact FPGA implementation of LED and PHOTON. In: International Conference in Cryptology in India, Springer (2014) 304–321
19. Black, J., Rogaway, P.: A block-cipher mode of operation for parallelizable message authentication. In: International Conference on the Theory and Applications of Cryptographic Techniques, Springer (2002) 384–397
20. Krovetz, T., Rogaway, P.: Ocb (v1. 1). (2016)
21. Minematsu, K.: AES-OTR v3.1. Technical report, (NEC Corporation, Japan (2016)
22. Homsirikamol, E., Diehl, W., Ferozpuri, A., Farahmand, F., Yalla, P., Kaps, J.P., Gaj, K.: CAESAR Hardware API. Cryptology ePrint Archive, Report 2016/626 (2016)
23. NIST: National Institute of Standards and Technology: Advanced Encryption Standard AES (2001)
24. El Maraghy, M., Hesham, S., El Ghany, M.A.A.: Real-time efficient fpga implementation of aes algorithm. In: SOC Conference (SOCC), 2013 IEEE 26th International, IEEE (2013) 203–208
25. Chaves, R., Kuzmanov, G., Vassiliadis, S., Sousa, L.: Reconfigurable memory based aes co-processor. In: Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International, IEEE (2006) 8–pp
26. Banik, S., Bogdanov, A., Regazzoni, F.: Atomic-AES v 2.0. Cryptology ePrint Archive, Report 2016/1005 (2016)
27. Ghaznavi, S., Gebotys, C., Elbaz, R.: Efficient technique for the FPGA implementation of the aes mixcolumns transformation. In: Reconfigurable Computing and FPGAs, 2009. ReConFig'09. International Conference on, IEEE (2009) 219–224
28. Resende, J.C., Chaves, R.: Compact dual block aes core on fpga for ccm protocol. In: Field Programmable Logic and Applications (FPL), 2015 25th International Conference on, IEEE (2015) 1–8
29. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED block cipher. In: International Workshop on Cryptographic Hardware and Embedded Systems, Springer (2011) 326–341