Lightweight ASIC Implementation of AEGIS-128

Anubhab Baksi, Vikramkumar Pudi, Swagata Mandal, Anupam Chattopadhyay
School of Computer Science and Engineering, Nanyang Technological University, Singapore
anubhab001@e.ntu.edu.sg, pudi@ntu.edu.sg, swagata.mandal@ntu.edu.sg, anupam@ntu.edu.sg

***Abstract*—In this paper, we study the problem of implementing the AEAD scheme, AEGIS-128, which is a finalist in the recently concluded competition, CAESAR. In order to achieve lightweight (least area) implementation, we first look into one round of AES encryption, which is a building block in this cipher. In this regard, we make use of the state-of-the-art implementation of AES in ASIC. We benchmark one round AES encryption (which is done for the first time) and later use it with AEGIS-128 to improve the optimized implementation reported (Inscrypt'14). Synthesis results show that our design requires 9.6% less area and reduces the power consumption by 95.3% (operating frequency is also reduced). Further, this concept can readily be applied to a variety of other ciphers.**

***Index Terms*— ASIC, optimization, encryption, authentication**

I. Introduction

Authenticated Encryption with Associated Data (AEAD) is a relatively recent concept in symmetric key cryptography where a cipher aims at providing confidentiality as well as authenticity. Given a message (referred to as, plaintext) and associated data; such a scheme encrypts the plaintext (confidentiality) and generates a tag depending on both the plaintext and the associated data (authentication). The sender (Alice) generates the ciphertext and the tag and sends them along with the associated data through an insecure channel (where the attacker, Eve, is active) to the recipient (Bob). Upon receiving the ciphertext and tag, bob generates the plaintext and the tag. If both the tags match, then the received ciphertext (hence the plaintext retrieved is original) and the associated data are considered not disturbed by Eve. Otherwise, Bob learns there is a disturbance caused by Eve; generates an invalid signal and discards the entire data received. For the transmission of plaintext, both confidentiality and authenticity is required; whereas only authenticity is required for the associated data. This is practical in scenarios like Internet packets, where the header is not confidential (acts as associated data) but the body is confidential (acts as plaintext).

To promote the research in this direction, a competition, named CAESAR[1] announces its finalists from 57 AEAD proposals. The cipher, AEGIS (currently version 1.1) [9] is among the 7 finalists (it is selected for high-performance applications). This cipher is based on the stream cipher design paradigm (XORing the plaintext with a pseudo-random string generated) to get the ciphertext, then the plaintext is fed into the state. Besides, it uses one round of AES-128 encryption (see description in Section II-B) for its state update operation.

This cipher gains popularity among researchers. Apart from being an academic interest (e.g., TIAOXIN, another CAESAR candidate [7] is inspired by it); it is also being used in practical applications (e.g., in a vehicular communication [8]).

In various cryptographic operations, it is often required to have an optimized hardware (ASIC/FPGA) or software (microcontroller/microcontroller) implementation. The reason is, tightly contained devices (such as sensors) are generally required to be equipped with cryptographic primitives. Hence a major flow of cryptographic research focuses on optimized design for a specific cipher. In this paper, we propose an ASIC implementation of AEGIS-128 (one recommended version) which consume least area compared to the other implementation reported. For the sake of simplicity, we henceforth refer to AEGIS-128 as AEGIS, unless otherwise mentioned. Our implementation can handle both the encryption & tag generation and decryption & verification modes. We mention that, our design strategy is equally applicable to other variants of AEGIS.

The ASIC flow has already been explored in the literature [3]. The smallest area implementation they are able to achieve is ≈ 18720 GE (Optimization AO_1 in Section 3.2; sometimes referred to as $A0_1$) for 65 nm technology with and power consumption 21.58 mW and clock frequency 1410 MHz. However, as we point out later on, there are scopes to improve this implementation. Also, unlike our case, they do not implement decryption & verification circuit. Nonetheless, we reduce area requirement by 9.6% compared to [3].

One key idea of our improved implementation is, utilize the "Atomic AES v2.0" developed by Banik *et al.* [2] (a follow-up work of [1]), which is the state-of-the-art lowest area implementation of AES-128 in ASIC. While the implementation of [2] is for original AES-128 (both encryption & decryption), we only need one round of AES-128 encryption for AEGIS implementation. Hence, we amend the design so as to meet our (reduced) requirement. As for the FPGA, only benchmark of AEGIS is reported so far [8], to the best of our knowledge. Another benchmarking for AEGIS-128L, a larger version, is reported by ATHENA[2].

Our Contributions

Our contributions in this paper are twofold. First, we benchmark one round AES-128 encryption (optimized area) for ASIC. This is a building block in quite a few ciphers, such

We thank the anonymous reviewers for their useful feedback. The first author likes to acknowledge the kind support of Prakash Dey (CU, India); Sachin Kumar, Sonu Jha, Arko Dutt, Mustafa Khairallah (NTU, Singapore).

[1] https://competitions.cr.yp.to/index.html

[2] https://cryptography.gmu.edu/athena/index.php?id=CAESAR_source_codes

2159-3477/18/$31.00 ©2018 IEEE
DOI 10.1109/ISVLSI.2018.00054

as, AEGIS [9] or DEOXYS [6] (both are finalists in CAESAR). To the best of our knowledge, despite of its frequent usage, it has never been benchmarked under any hardware platform. Our results are shown in Section IV-A.

Second, we apply this area optimized one round AES-128 encryption to the AEAD cipher AEGIS. This helps us to reduce the ASIC area and power consumption significantly. Compared to the design in [3], ours requires less area (9.6%) and very less power (95.3%), however the operating frequency in our case is also reduced.

Paper Organization

The rest of the paper is organized as follows. Section II describes all the components of the AEGIS cipher. In Section III, we describe the ideas used in the previous works (one on AEGIS ASIC implementation, another on AES-128 encryption). Following these, we present how we implement our concept on improved ASIC version of AEGIS. The results we obtain are presented in Section IV. Finally, Section V concludes the paper, citing interesting future works.

II. Description of AEGIS

A. Overview

Before proceeding further, here we present a quick overview of the AEGIS. It has the following parameters: 128-bit key, K; 128-bit initialization vector, IV; and 640-bit state. The state size is reduced from AEGIS-128L (which has a state size of 1024-bit), and it is suitable for high performance applications. The lengths of associated data, AD and the plaintext P (and hence the length of the ciphertext, C) are to be less than 2^{64}-bits. The recommended length of the authentication tag is 128-bit; although the design allows to generate tag of an arbitrary length ≤ 128. In AEGIS, both $|AD|$ and $|PT|$ are communicated to Bob.

B. One Round AES-128 Encryption

It is common to use one round of AES-128[3] encryption in AEAD design. In this part, we briefly describe the original AES-128 encryption. The cipher takes 128-bit input (plaintext) and 128-bit key, and generates 128-bit output (ciphertext).

The input is rearranged into a 4×4 matrix of bytes (referred to as the 'state'), which is normally accessed in the column-major way. The key is XORed with the state. The key is also passed to the 'key scheduling' algorithm, which generates 10 round keys – each round key is of 128-bits, and the process is reversible (given any round key, it is possible to recover the key).

The state undergoes 10 rounds of modifications. The first 9 rounds are identical, each consists of 4 steps in order: 'SubBytes', 'ShiftRow', 'MixColumn' and 'AddRoundKey'. The last (10^{th}) round is little different, it only has SubBytes, ShiftRow and AddRoundKey (misses MixColumn).

The AddRoundKey step XORs the corresponding round key to the state. This step is performed at the very beginning (before 1^{st} round), where the key is XORed with the state (= plaintext). The rest steps are described below.

- SubBytes. This step contains the only non-linear operation in the cipher. Here, the input is converted with the help of a look-up table (known as, 'SBox').
- ShiftRow. This step affects the state row-wise. Each row of state is rotated by a fixed number of rotations.
- MixColumn. This step operates on the state column-wise. Each column of the state is considered a 4-dimensional vector, each element of which belongs to GF(2^8). A 4×4 matrix, M (whose elements are also in GF(2^8)), is multiplied to each column. The result thus obtained is used to substitute the old contents of the column. This process is repeated for all 4 columns.

The one round AES-128 encryption is a simplified version of the original which can use any of the one round from 1 to 9. Here, the 128-bit plaintext, after arranging to a 4×4 matrix of bytes, undergoes SubBytes, followed by ShiftRow, followed by MixColumn. Finally the key is XORed. Figure 1 shows one round AES-128 encryption. Henceforth, the first three combined steps are denoted by the notation $R(\cdot)$; so, $R(p) = \text{MixColumn}(\text{ShiftRow}(\text{SubBytes}(p)))$.

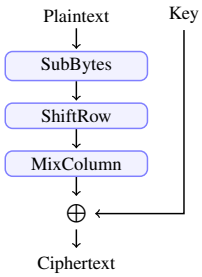

Fig. 1: One round AES-128 encryption

C. AEGIS State Update

The state of AEGIS consists of five 16-byte (= 128-bit) registers, denoted by S_0, S_1, S_2, S_3 and S_4, totaling 640-bits. Another 128-bit register M is also used.

Let us denote the j^{th} register at i^{th} round by $S_{i,j}$, $j = 0(1)4$, and the content of the message register M as M_i. Then, the StateUpdate128(S_i, M_i) operation (which updates all 5 registers) is given as (recall definition of R from Section II-B):

$$\begin{aligned} S_{i+1,0} &\leftarrow R(S_{i,4}) \oplus S_{i,0} \oplus M_i; \\ S_{i+1,1} &\leftarrow R(S_{i,0}) \oplus S_{i,1}; \\ S_{i+1,2} &\leftarrow R(S_{i,1}) \oplus S_{i,2}; \\ S_{i+1,3} &\leftarrow R(S_{i,2}) \oplus S_{i,3}; \\ S_{i+1,4} &\leftarrow R(S_{i,3}) \oplus S_{i,4}. \end{aligned}$$

Figure 2 shows the structure of AEGIS state update operation.

[3] https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

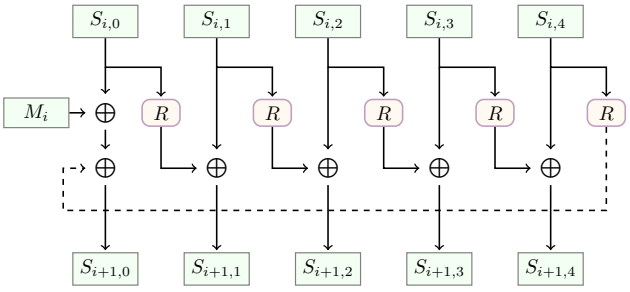

FIG. 2: AEGIS state update function ($\texttt{StateUpdate128}(S_i, M_i)$)

D. Initialization

During the initialization, the key (K) and initialization vector (IV) are loaded in the state, then the state is updated 10 rounds. The detailed procedure is as given below. Here const_0 and const_1 are two pre-defined 128-bit constants, $\text{const}_0 \coloneqq$ `000101020305080d1522375990e97962` and $\text{const}_1 \coloneqq$ `db3d18556dc22ff12011314273b528dd`.

1) Key, IV are loaded at first:

$$\begin{aligned} S_{-10,0} &\leftarrow K \oplus IV; \\ S_{-10,1} &\leftarrow \text{const}_1; \\ S_{-10,2} &\leftarrow \text{const}_0; \\ S_{-10,3} &\leftarrow K \oplus \text{const}_0; \\ S_{-10,4} &\leftarrow K \oplus \text{const}_1. \end{aligned}$$

2) For $i = -5(1) - 1$:

$$\begin{aligned} M_{2i} &\leftarrow K; \\ M_{2i+1} &\leftarrow K \oplus IV. \end{aligned}$$

3) For $i = -10(1) - 1$:

$$S_{i+1} \leftarrow \texttt{StateUpdate128}(S_i, M_i).$$

E. Associated Data Processing

After key and IV are loaded to the state, associated data (AD) is loaded. If $|AD| = 0$, then this step is skipped. If $|AD|$ is not a multiple of 128, then zero's are padded to AD to make its length a multiple of 128.

For $i = 0(1)u - 1$, the state update is done as given below, here $u = \lceil |AD|/128 \rceil$. We denote the i^{th} 128-bit block of AD by AD_i.

$$S_{i+1} \leftarrow \texttt{StateUpdate128}(S_i, AD_i).$$

F. Encryption

The encryption operation follows, where the plaintext P is used to update state, and to produce the ciphertext C. If $|P| = 0$, then this step is skipped. If $|P|$ is not a multiple of 128, then zero's are padded so that $|P|$ becomes a multiple of 128.

For $i = 0(1)v - 1$, the state update is done as given below, $v = \lceil |P|/128 \rceil$. We use the notation, P_i (resp. C_i) to denote the i^{th} 128-bit block of the plaintext P (resp. ciphertext C).

$$\begin{aligned} C_i &\leftarrow P_i \oplus S_{u+i,1} \oplus S_{u+i,4} \oplus (S_{u+i,2} \cdot S_{u+i,3}); \\ S_{u+i+1} &\leftarrow \texttt{StateUpdate128}(S_{u+i}, P_i). \end{aligned}$$

G. Finalization

After encryption, the authentication tag (T) is generated with seven more `StateUpdate128` operations.

Let, $q = S_{u+v,3} \oplus (|AD| \;||\; |P|)$, where both $|AD|$ and $|P|$ are represented as 64-bit values. For $i = u + v(1)u + v + 6$, the following state update is done:

$$S_{i+1} \leftarrow \texttt{StateUpdate128}(S_{u+i}, q).$$

The authentication tag, T, is the first t-bits ($0 < t \leq 128$, recommended value of t is 128) of $T' = \bigoplus_{i=0}^{4} S_{u+v+7,i}$

H. Decryption & Verification

For decryption, the procedure is similar. It first does the initialization and processing of associated data steps. Next, the ciphertext C is decrypted from the plaintext P. If $|C|$ is not a multiple of 128, then zero's are padded to make it a multiple of 128.

For $i = 0(1)v - 1$, the decryption and state update are done as follows:

$$\begin{aligned} P_i &\leftarrow C_i \oplus S_{u+i,1} \oplus S_{u+i,4} \oplus (S_{u+i,2} \cdot S_{u+i,3}); \\ S_{u+i+1} &\leftarrow \texttt{StateUpdate128}(S_{u+i}, P_i). \end{aligned}$$

The finalization step is same as its namesake in encryption. If the tag verification fails, then the ciphertext and the newly generated tags are not given as output.

III. Optimization: Minimizing ASIC Area

In this section, we elaborate our optimization concept that we use here to minimize the ASIC area. First, we describe the concept used in [3]; and then we explain how the ASIC area is minimized in one round `AES-128` encryption [2] (and [1]). Later, we explain the design considerations we incorporated.

A. Basic Optimization on AEGIS

The design objectives in [3] are twofold — area and throughput optimization. As throughput optimization is out of scope for our work, we only focus at the area optimized implementation. More specifically, we consider the smallest area implementation (18.72 kGE), AO_1 (or, A0_1), which also consumes least power (21.58 mW).

The authors, at first, design a base implementation with an ordinary version of `AES-128` one round encryption. Following this, they adopt the optimized SBox implementation from [5]. However, the SBox implementation in [5] deals with masking, which is a countermeasure against side-channel attacks. This countermeasure works by incorporating random values, which are XORed or ANDed with the internal components. Naturally, the area requirement for this implementation is considerably higher than its non-masked counterpart, which is given in [4].

B. Optimization on AES-128 Encryption

The design of AES-128 in [2] is an improvement on top of the design in [1]. Hence, for the sake of conciseness, we only describe the basic design choices of Atomic AES 2.0 (encryption only) in a nutshell, as the detailed description can be found in [2].

It uses 16 registers (arranged in a 4×4 matrix), each of size 8-bits to store intermediate states and round keys separately (32 registers in total). Few registers are implemented with ordinary flip-flops, while the others are implemented with scan flip-flops. Ordinary flip-flops allow data movement in horizontal direction only, while scan flip-flops allow data to move both horizontally and vertically. The plaintext and the key are loaded serially in first 16 clock cycles. It produces the output after 246 clock cycles.

C. Improved Optimization on AEGIS (Ours)

Our implementation of R incorporates designs ideas from AES-128 implementation (Section III-B) with a few modifications. The main change from the aforementioned implementation is, we skip the key scheduling function and few other components that are used only in decryption process. Also, we need lesser rounds. This helps to reduce the area. Further, we also skip the step where the key is XORed with the state (before the 1st round operation).

Once the design of R is completed, we focus on the main architecture of AEGIS cipher. Since we want to minimize area requirement, we instantiate R only once, which means we access it sequentially; rather than instantiating it five times and access them parallelly. In this case, one astute observation regarding the StateUpdate128 operation is that, backing-up state registers $(S_0, \ldots, S_3)$ is required (S_4 is not needed to be backed-up). This is because, the update of one register depends on old content of another register. For example, when computing $S_{i+1,1}$; $S_{i,0}$ is needed (refer to Section II-C). Now, $S_{i,0}$ is replaced by $S_{i+1,0}$ already; so $S_{i,0}$ is to be backed-up in another register. In the naive approach, four back-up registers are required.

However, using an optimization, it is possible to reduce the number of back-up registers down to two. With the five 128-bit registers, $S_0, \ldots, S_4$; 128-bit message register M; consider the 128-bit back-up registers B^1 and B^2. The subscript i denotes the contents of $S_0, \ldots, S_4$ at i^{th} round. Here B^1, B^2 are updated five times within one StateUpdate128; we call each update as a step; so B^i_j denotes the content of B^i at j^{th} step, $i = 1, 2$. The following sequence of operations show how StateUpdate128 can be done:

$$
\begin{aligned}
B^1_{5i} &\leftarrow R(S_{i,4}) \oplus M_i; \\
B^2_{5i} &\leftarrow S_{i,0}; \\
S_{i+1,0} &\leftarrow B^1_{5i} \oplus B^2_{5i}; \\
B^1_{5i+1} &\leftarrow R(S_{i+1,0} \oplus B^1_{5i}); \\
B^2_{5i+1} &\leftarrow S_{i,1}; \\
S_{i+1,1} &\leftarrow B^1_{5i+1} \oplus B^2_{5i+1}; \\
B^1_{5i+2} &\leftarrow R(S_{i+1,1} \oplus B^1_{5i+1}); \\
B^2_{5i+2} &\leftarrow S_{i,2}; \\
S_{i+1,2} &\leftarrow B^1_{5i+2} \oplus B^2_{5i+2}; \\
B^1_{5i+3} &\leftarrow R(S_{i+1,2} \oplus B^1_{5i+2}); \\
B^2_{5i+3} &\leftarrow S_{i,3}; \\
S_{i+1,3} &\leftarrow B^1_{5i+3} \oplus B^2_{5i+3}; \\
B^1_{5i+4} &\leftarrow R(S_{i+1,3} \oplus B^1_{5i+3}); \\
B^2_{5i+4} &\leftarrow S_{i,4}; \\
S_{i+1,4} &\leftarrow B^1_{5i+4} \oplus B^2_{5i+4}.
\end{aligned}
$$

IV. Results

In this Section, we present synthesis results for AES-128 one round encryption and our AEGIS design. We use VHDL for the coding part; and then Synopsys Design Compiler version H-2013.03-SP1, using the 65 nm CMOS TSMC technology library for the synthesis and evaluation. The area is reported in terms of kilo Gate-Equivalent (kGE).

A. One Round AES-128 Encryption

As mentioned previously, we use AES-128 one round encryption as a black-box, the top-level architecture of which is shown in Figure 3. The usage of the input-output signals is straightforward.

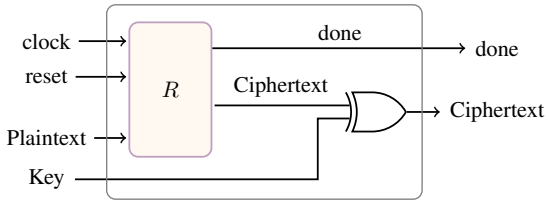

FIG. 3: Top level design of AES-128 1 round encryption

Here we present the ASIC synthesis results for the component R of AES-128 one round encryption. Our design requires 2.21 kGE for implementation, achieved maximum operational frequency as 355 MHz, 0.45 mW power and takes 21 clock cycles to generate the output. As a comparison, it requires 24.6% and 9% less area compared to Atomic AES and Atomic AES v2.0, respectively.

B. AEGIS

1) Implementation Overview

Here we outline the basic design concept we use for AEGIS core, which encompasses the key components. The top level description is given in Figure 4. The use of the signals clock, reset and start are straightforward. The mode input identifies the mode of operation, i.e., it is low for encryption & tag generation; and high for decryption & verification. The next six input lines, S_0_in, $\ldots, S_4$_in, M_in receives inputs for the corresponding 128-bit registers bit-by-bit. The following three input lines indicate which input is being fed to M: AD_active, is high for AD blocks; P_active for plaintext blocks for encryption & tag generation (ciphertext blocks for decryption & authentication) and T_active for $|AD| \,||\, |PT|$ (it is high for the first round

when $|AD|$ || $|PT|$ is fed; refer to Section II-G for more details) for the particular round. For example, if plaintext is 2^{32}-bits long, P_active will be high for $2^{32-7} = 2^{25}$ rounds, as each round processes 2^7 bits. As for the outputs, C_out and T_out give the ciphertext (during encryption & tag generation) or the plaintext (during decryption & verification) and tag bit-by-bit; and the done signal is high twice to indicate whether the generated ciphertext/plaintext and tag are valid, respectively. Note that, this core has to be implemented within a wrapper. This wrapper will control all the input signals (e.g., send plaintext blocks as bit-by-bit); as well as manage the output signals. In case of decryption & verification, this wrapper will be responsible to check the tag is matched and do the follow-up steps (see Section II-H).

Next, we describe the architecture in short. The basic architecture is pictorially presented in Figure 5. The input signal clock controls the signal round_counter, which is used to update other registers. The registers, $S_0, \ldots, S_4, M, B^1, B^2$ are described earlier. The 128-bit register C is used to store the encrypted ciphertext (during encryption & tag generation) or the decrypted plaintext (during decryption & verification), which is determined by the mode input. The 128-bit register T stores the tag. As mentioned already, R denotes the one round AES-128 encryption. Our implementation takes 38 clock cycles to update one register (which includes, operating on R and accessing B^1, B^2), so the StateUpdate128 operation takes $38 \times 5 = 190$ clock cycles. The *done control* part controls when the output signal done will be high or when it will be low.

One crucial component in the design is the 238-bit round_counter. This signal controls the inputs/outputs across the circuit. For example, the 1-bit signals, S_0_in, ..., S_4_in, M_in are demultiplexed to the corresponding 128-bit signals; and the 128-bit signals, C, T are multiplexed to 1-bit C_out, T_out; respectively.

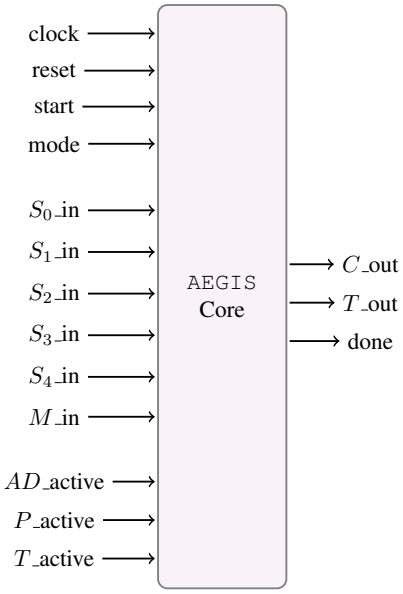

FIG. 4: Top level design of AEGIS core

The rationale for keeping the length of round_counter as 238 bits is explained below:

- 1-bit to indicate whether data are being loaded to the registers or StateUpdate128 is in operation.
- 8-bits to keep track of current status of data loading/ StateUpdate128. Note that, to load 128-bit inputs, we need 7-bits; but since StateUpdate128 takes 190 clock cycles, we use 8-bits.
- 57-bits to keep track of round number during AD loading. Since $|AD| < 2^{64}$, and 2^7 bits are loaded in one round, we need at most $64 - 7 = 57$ bits as the counter.
- Similarly, 57-bits to keep track of round number during P loading.
- 114-bits to store the round number when T_in is high for the last time, say, at w^{th} round. We need this, as we now have to run StateUpdate128 for six more rounds, i.e., till the $(w+6)^{\text{th}}$ round.
- 1 more bit is needed so that the round counter can go six more rounds.

2) *Synthesis Results*

In Table I, we present the ASIC synthesis results for AEGIS and compared with the existing design in AEGIS [3]. The AEGIS design requires only 16.92 kGEs for implementation and achieved maximum operational frequency as 208 MHz as given in Table I. Our AEGIS implementation requires 1.8 less kGE compared to the design in [3] (which also uses 65 nm technology), and also requires very less power compared to the design (but the operating frequency is also significantly reduced). In mathematical terms, our implementation requires 9.6% less area and reduces the power consumption by 95.3% compared to the design AO_1. The area reduction in our case is achieved by optimizing poor design choices. We would like to reiterate that, our implementation contains both encryption & tag generation as well as decryption & verification circuits; whereas [3] only contains encryption & tag generation circuit.

TABLE I: ASIC synthesis results for AEGIS

Design	Area (kGE)	Frequency (MHz)	Power (mW)
AEGIS (Ours)	16.92	208	1.007
AEGIS (AO_1, [3])	18.72	1410	21.58

V. CONCLUSION

Our work in this paper deals with implementation of one round AES-128 encryption (in ASIC, with minimum area), and later apply this primitive to the AEAD cipher, AEGIS-128. The former result is reported for the first time, whereas the latter result is improved from a previous work. An interesting follow-up work can be to see how this primitive can be applied to other AEADs where AES-128 encryption is used as a building block, such as TIAOXIN or DEOXYS. Besides, similar optimization on variants of AEGIS (other than AEGIS-128; such as AEGIS-128L) can also be performed.

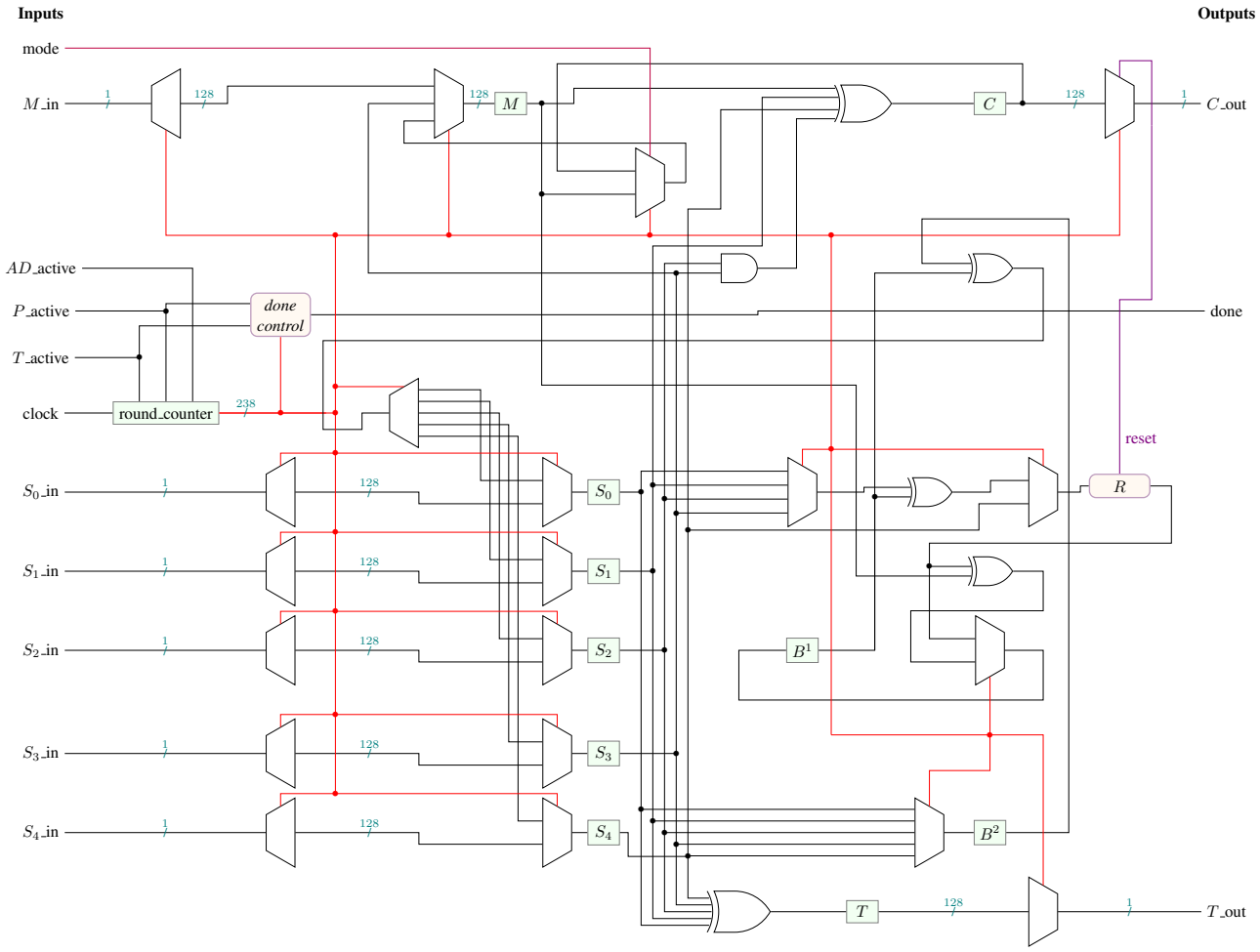

FIG. 5: Architecture of AEGIS core

REFERENCES

[1] Banik, S., Bogdanov, A., Regazzoni, F.: Atomic-aes: A compact implementation of the AES encryption/decryption core. In: Progress in Cryptology - INDOCRYPT 2016 - 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings. (2016) 173–190

[2] Banik, S., Bogdanov, A., Regazzoni, F.: Atomic-aes v2.0. Cryptology ePrint Archive, Report 2016/1005 (2016) https://eprint.iacr.org/2016/1005.

[3] Bhattacharjee, D., Chattopadhyay, A.: Efficient hardware accelerator for AEGIS-128 authenticated encryption. In: Information Security and Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers. (2014) 385–402

[4] Canright, D.: A very compact s-box for aes. In Rao, J.R., Sunar, B., eds.: Cryptographic Hardware and Embedded Systems – CHES 2005, Berlin, Heidelberg, Springer Berlin Heidelberg (2005) 441–455

[5] Canright, D., Batina, L.: A very compact "perfectly masked" s-box for AES. In: Applied Cryptography and Network Security, 6th International Conference, ACNS 2008, New York, NY, USA, June 3-6, 2008. Proceedings. (2008) 446–459

[6] Jean, J., Nikolić, I., Peyrin, T., Seurin, Y.: Deoxys v1.41. https://competitions.cr.yp.to/round3/deoxysv141.pdf Accessed: 2018-02-27.

[7] Nikolić, I.: Tiaoxin – 346 (v2.1). https://competitions.cr.yp.to/round3/tiaoxinv21.pdf Accessed: 2018-02-27.

[8] Wang, Y., An, J., Ha, Y.: Unified data authenticated encryption for vehicular communication. In: 2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS). (Oct 2016) 1–4

[9] Wu, H., Preneel, B.: AEGIS: A Fast Authenticated Encryption Algorithm (v1.1). https://competitions.cr.yp.to/round3/aegisv11.pdf Accessed: 2018-01-22.