Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186–5140
Volume 5, Number 1, pages 26–33, January 2016

Optimization of Hardware Implementations with High-Level Synthesis of Authenticated Encryption

Makoto Kotegawa, Keisuke Iwai, Hidema Tanaka and Takakazu Kurokawa
National Defense Academy of Japan
Yokosuka, Japan
Email: em53037@nda.ac.jp

***Abstract*—Competition for Authenticated Encryption: Security, Applicability, and Robustness(CAESAR) is carried out as development and an evaluation of new authenticated encryption. We performed hardware implementations with VIVADO High-Level Synthesis which is a tool of Xilinx. This tool is used with some "directives" for optimization. This paper shows various optimization techniques on the point of speed, area size and the clock frequency.**

***Index Terms*—AES, AES-OTR, Authenticated Encryption, Hardware implementation, High-Level Synthesis, POET, SILC, ZYNQ-7000.**

I. Introduction

In the general encrypted communication, detection of the falsification is difficult. If user decrypt and use falsification message unawarely, a security problem will be occurred. Use of authenticated encryption is one of the way to solve this problem.

Authenticated encryption has 3∼4 inputs as plain text, secret key, nonce and associated data, moreover, 2 kind of outputs as cipher text and tag. Tag is fixed length data, which depends on plain text or cipher text, and its length is defined by their specification. For example, let us consider a case of communication between 2 people with authenticated encryption. The sender makes cipher text and tag, and sends them to the recipient. The recipient makes a tag from the received cipher text. When cipher text is tampered by someone, sender's tag should be different from recipient's tag. In this way, detection of the manipulation will be possible by using authenticated encryption. This technique attracts attention in not only use of computers but also embedded systems because of their wide use in global network.

CAESAR(Competition for Authenticated Encryption: Security, Applicability, and Robustness) is one of the main action for authenticated encryptions. CAESAR invites public participation of authenticated encryptions and adopt their next-generation[1]. At the start of CAESAR, 47 algorithms were applied. As the first selection of CAESAR, 29 algorithms passed in the summer 2014, and they were proceeded the second selection. Among 15 algorithms adopt AES for their primitives as shown in Table. I. Selection meetings are held in Direction In Authenticated Ciphers(DIAC) in every summer. The adoption will be in December 2017 after the last selection meeting at DIAC.

As for the implementation, both software and hardware are assumed because authenticated encryption is necessary for not only PCs but also embedded systems. We paid attention to hardware implementations and checked a recent tendency including two representative methods using ASIC(Application Specific Integrated Circuit) or FPGA(Field-Programmable Gate Array). Although ASIC and FPGA have good points and bad points, we decided to implement authenticated encryption on FPGA which can be updated remotely. This is because most embedded systems are designed to connected to network(e.g Internet)[2], [3].

In addition, for software developers, High-Level synthesis(HLS) tools enable us to convert from a source code written by high-level language(e.g. C language) to Register Transfer Level(RTL) . It enables us to shorten hardware development period and simplify the designing process to for the person who does not know the hardware language enough. From this characteristic, HLS is attracted attention at the present. Thus, we selected hardware implementations with HLS[4], [5].

TABLE I
Candidates of CAESAR in second round and their classification

	Using AES for the primitive	NOT using AES for the primitive
Nonce Based	AES-AEGIS, **AES-OTR**, CLOC, Deoxys*, Joltik*, OCB, **SILC**, Tiaoxin	ACORN, KetjeKeyak, NORX, PRIMATEs-GIBBON, PRIMATEs-HANUMAN, SCREAM, STRIBOB, π-cipher,
Nonce-misuse Resistance	**AES-COPA**, AES-JAMBU, AEZ, Deoxys*, ELmD, Joltik*, PAEQ, **POET**, SHELL	Ascon, HS1-SIV, ICEPOLE, Minalpher, MORUS, OMD

* The algorithm which can use both NB and NR.

II. CAESAR

A. Candidates and the classification

At present, 29 candidates are facing the second round selection in CAESAR. Table. I shows these candidates. Candidates of CAESAR are classified by using AES for their primitives

or not by presented by F.Abed[6]. Furthermore, these candidates are also classified as Nonce Based or Nonce-misuse Resistance. Here nonce is a kind of one-time random number.

Cryptographic primitives, using AES are acceptable to the implementation side. This is because AES is used as a representative of block cipher and many high speed implementations has been already presented[2]. In addition, high speed processing using exclusive instructions for AES is possible. These instructions are defined by representable processors such as Intel Core series or ARM Cortex series. Authenticated encryption using AES for the primitives have big advantages for speedup, because Intel Core and ARM Cortex processors are used in both areas of PC and embedded systems.

Next, we describe the difference of Nonce Based and Nonce-misuse Resistance. Here, we use "block" in a meaning "128 bits block" afterward. The followings are their characteristics.

Nonce Based(NB)
: Requiring nonce. The number of primitives used in Nonce Based is less than Nonce-misuse Resistance to encrypt a block of plain text.

Nonce-misuse Resistance(NR)
: Requiring no nonce. The number of primitives used in Nonce-misuse Resistance is more than Nonce Based when encrypting a block of plain text.

Nonce based has an advantage of processing speed. On the other hand, Nonce-misuse Resistance has an advantage that does not have to share nonce in advance, so that there is no weakness for nonce. Here we follow this classification.

B. Former studies for implementation

There is a representative former study[7] that implemented authenticated encryptions submitted to CAESAR, and evaluated their processing speed. This paper selected 12 candidates of CAESAR using AES for their primitive, and implemented them as software, then measured processing speed. On the experiment environment, they assumed communication of the Internet between PCs, and the features are summarized as follows.

- Assumed the use of an authenticated encryption for communication in the Internet, setting plain text length to 128$\sim$2048 Bytes.
- Compared and evaluated each candidates, dividing them into Nonce Based/Nonce-misuse Resistance.
- Strove for speed-up by using AVX[8][1] and AES-NI[9][2] supported by Haswell architecture CPU[3].

In consequence of the experiment, the fastest candidate was AES-OTR in NB, and AES-COPA in NR For our experiment, we chose authenticated encryption algorithms.

[1]Intel Advanced Vector eXtensions: It is extended instruction set which Intel developed, and can operation width of up to 256bit.

[2]AES New Instruction: The instruction set that is supported for the purpose high-speed coding and decoding of AES by Intel CPU

[3]The latest Intel CPU architecture as of 2014.

C. Choice of algorithms for optimization

For our experiment, we chose 4 authenticated encryption algorithms out of 12 algorithms. At first, we chose AES-OTR in Nonce Based and AES-COPA in Nonce-misuse Resistance. Because their processing speed became the fastest in the [7].

Next, we chose POET in Nonce-misuse Resistance, which has universal hash function in encrypt/decrypt algorithm. Universal hash function is defined as 4-round AES to the first, and Full-round AES to the second in its specification. Using Full-round AES for universal hash function was implemented [7], while we implemented POET using 4-round AES for its universal hash function from the viewpoint of speed-up.

In addition, we chose SILC which is Nonce Based authenticated encryption. SILC is inferior to AES-OTR on the stand point at processing speed because SILC needs more AES calls in tag generating algorithm than AES-OTR. In contrast, SILC is superior of implementation area because SILC was designed for the purpose of small implementation area. We thought that not only the processing speed but also the implementation area are essential because the latter is usually important factor plays on embedded systems. Hence we chose SILC for evaluation. Outline of these algorithms are summarized below and Fig. 1. The left side at Fig. 1 is encryption process and the right side is tag generating process.

AES-OTR(Nonce Based)[10]
: AES-OTR has Feistel structure and uses two variables δ and $L(=4\delta)$ which depend on Nonce. In encryption process and decryption process, parallel processing is possible in every two blocks. The number of AES calls depends on the length of plain text and cipher text in encryption or decryption, and is independent in the tag generating process. AES-OTR can generate tag with an AES call.

SILC(Nonce Based)[11]
: CFB mode is applied to SILC which does not use intermediate variable. Parallel processing is impossible in encryption process, while it is possible in decryption process. In encryption process, the number of AES calls depend on the length of plain text which is the same as OTR. However, SILC also depends on cipher text length in tag generating process. Consequently, SILC is inferior to AES-OTR in processing speed when plain text is long [4].
By contrast, SILC is designed to reduce the number of functions and variables for embedded system. As a result, it is expected that SILC has advantage of implementation area.

AES-COPA(Nonce-misuse Resistance)[12]
: In encryption process, the number of AES calls is 2

[4]The number of executing AES in SILC is defined as follows by specifications.
|A|: Associated Data length[Byte], |M|: plain text length[Byte]
n: length per a block [Byte].
AES-OTR: $\lceil |A|/n \rceil + \lceil |M|/n \rceil + 3$ (one call can be preprocessed)
SILC : $\lceil |A|/n \rceil + 2\lceil |M|/n \rceil + 2 + \lceil |N|/n \rceil$

TABLE II
FEATURES OF FPGA AND ASIC

Device	FPGA	ASIC
Cost	high	low
Need of prototype	no	yes
Development period	fast	not fast
Field reprogramability	yes	no

every plain text block. The number of call AES is 2 and it is independent of plain text length. Parallel processing is possible and variable $L(= E_k(0))$ is used. In generation tag process, the number of AES calls is 2 and it is independent of plain text length.

POET[13]

In POET encryption process, it needs an AES call and 2 Universal Hash Function(UHF) calls E_k(4-Round or full-Round AES).POET uses two variables X and Y which are generated by Associated Data. About UHF, it is defined to use 4-round AES in the first place or full-round AES in the second place. In the former case, the number of AES call in POET is less than AES-COPA. On the other hand, POET uses three kinds of secret key those are generated by AES calls. Hence, when the update of secret keys is frequent, processing speed is expected to be down and implementation area grows large.

III. HARDWARE IMPLEMENTATION

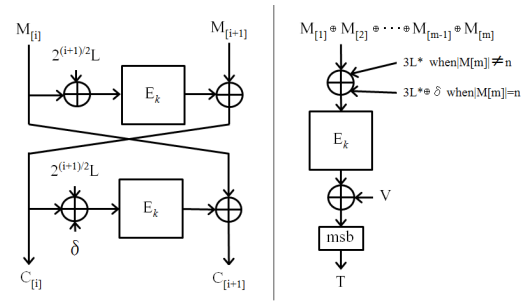
A. Comparison between FPGA and ASIC

In hardware implementation, there are 2 kind methods which use FPGA or ASIC. We show the comparison of their features in Table. II[3].

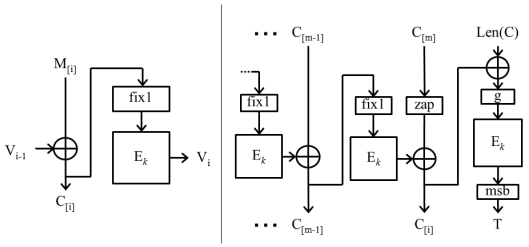
ASIC is used widely in implementation in mass production. In the meantime, FPGA is unsuitable for implementation the stand point of their cost. Although FPGA has a great advantage in today's network society, as its remote reprogramability.

Let us show the while life cycle of products which has computer or micro-computer. They are often updated until the end of their life cycle and the update becomes frequent like PCs. In addition, frequent update is also common in embedded systems. We can find typical examples in developing of control systems which are represented by engine control and Advanced Driver Assistance Systems. If a developer tries to update ASIC, it is necessary to re-product ASIC from the beginning. By contrast, when try to update FPGA, we only connect FPGA to network and download bit-stream files which defines the circuit in FPGA. This difference by remote reprogramability is not negligible.

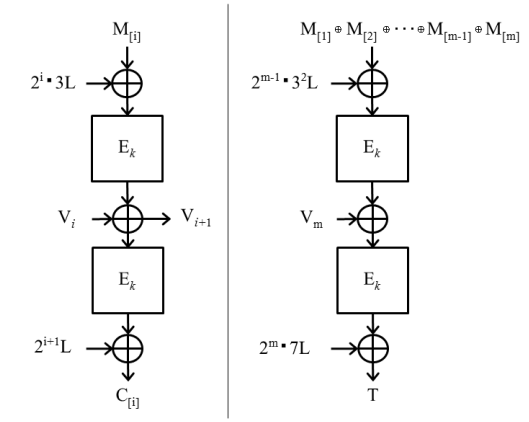
Thus it is important that hardware has remote reprogramability or not. We assume that FPGA is suitable for developed embedded system than ASIC.



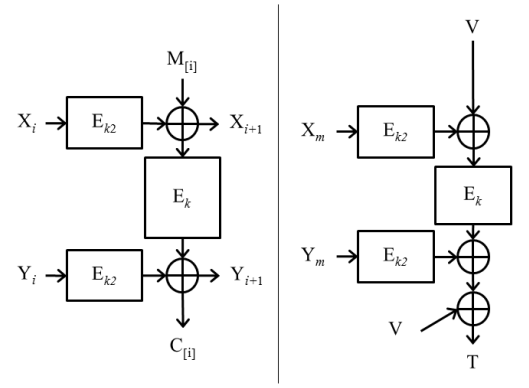
(a) OTR



(b) SILC



(c) COPA



(d) POET

M:plain text, C:cipher text, T:Tag, Ek:AES, V:Hash value

Fig. 1. Outlines of the authenticated encryptions chose for our optimization.

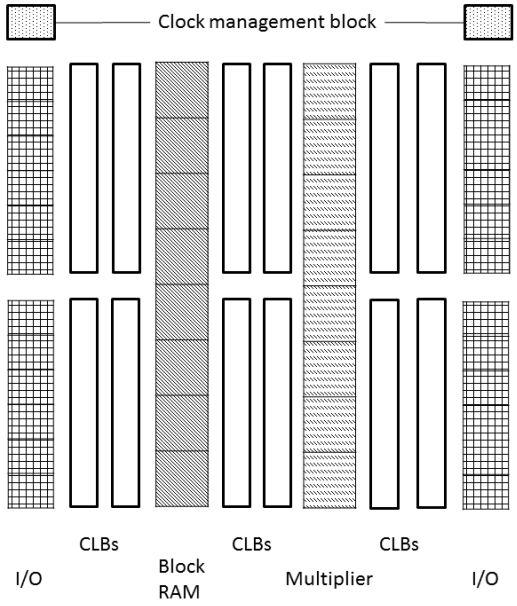


Fig. 2. Example architecture diagram of FPGA[16]

B. FPGA

1) Outline: FPGA is a type of programmable logic device, Just like the name, it is enable us to build custom circuit by programming. Here, we explain constitution of FPGA. The main part constituting FPGA is Configurable Logic Block(CLB), PLL block, block RAM, multiplier and I/O element as shown in Fig. 2). The followings are their descriptions[15], [16].

Configurable Logic Block(CLB)
: CLB is the core parts of FPGA, and consists of Look Up Table(LUT), multiplexers, resistor and storage element. It is possible to constitute memory by LUT and sequential circuit by register, moreover it can be made logic circuit by controlled LUT and register. FPGA have switches for controlling of LUT and register and we can configure switches by reading the configuration data which called bit-stream file. According to this mechanism, it is possible to design circuit by programming.

Clock management block
: Clock management block generates and supplies clock wave of various kinds of frequency and high precision like a standard clock in FPGA.

Block RAM and Multiplier
: It is possible to use CLB as a memory and multiplier, however it reduces the efficiency of using the CLB. That' why, usually using them when it is required to implement memory and multiplier in FPGA.

I/O element
: I/O element in FPGA is also programmable to accommodate various interface because it is necessary to match the electrical characteristics of the signal of the in/output.

TABLE III
SPECIFICATION OF ZYNQ-7000 XC7Z020[14]

CPU	ARM Cortex-A9 32bit processor 886MHz	
F P G A	Programmable Logic Cells	85K Cells
	Look-Up Tables	53200
	Flip-flops	106400
	Block RAM	560KB
	Logic Slice	13300

By these characteristics, hardware programming is possible. In addition, hardware developer is able to build custom circuit repeatedly using the same device(FPGA) without re-product devices from the beginning like ASIC.

2) ZYNQ-7000: We used ZYNQ-7000 XC7Z020 which is System-On-a-Chip(SOC) including Artix-7 FPGA and Dual ARM Cortex-A9 CPU[14]. The specification of ZYNQ-7000 XC7Z020 is summarized as follows and shown in Table. *III*

It is possible to program flexibly each I/O, software and hardware because ZYNQ-7000 have both of FPGA and CPU. In part of FPGA, ZYNQ have 6650 Configurable Logic Blocks(CLB)[16]. A CLB in ZYNQ-7000 have 2 Slices, 8 LUTs, 16 Flip-Flops as register. 2 Carry Chains, 128 bits Shift Registers and 256 bits Distributed RAM. The followings and Fig. 3 are description on CLB.

Logic slice
: A CLB in ZYNQ-7000 has different types of Slice called SLICEM and SLICEL. Both of them have 4 LUTs, 8 Flip-Flops, multiplexers, Carry logic as shown in Fig. 3. In addition, SLICEM supports two additional functions that storing data using distributed RAM and shift registers, It keeps all the basic functions necessary to use FPGA in Slice.

Carry logic
: Carry logic is dedicating fast lookahead carry. It provides to perform fast arithmetic addition and subtraction in slice.

Shift Registers in SLICEM
: A SLICEM can be configured as a 32-bit shift register without using the flip-flops. So, each LUT can delay serial data from 1 to 32 clock cycles. Furthermore, SLICEM can be configured as a 128-bit shift register because SLICEM has 4 LUTs.

Distributed RAM in SLICEM
: The LUTs in SLICEM can be implemented as a RAM resource called a distributed RAM element. Distributed RAM using LUTs in SLICEM can be combined in various ways to store large amount of data, and also be configured some kind of RAM such as single-port or dual-port. Moreover, Distributed

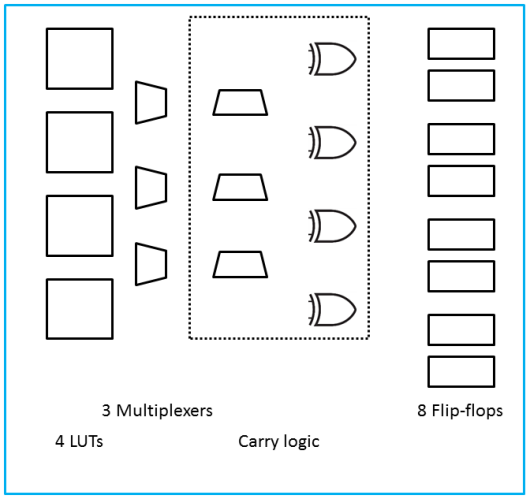


Fig. 3. Slice in ZYNQ-7000[16]

RAM has an advantage that the memory access is faster than using block memory, and also has a disadvantage that the implementation area becames lager.

In our implementation of authenticated encryption, we adopt the number of slices as a standard to compare the implementation area.

C. High-Level Synthesis

A general way for the implementations of circuits on FPGA requires to describe hand code in Register Transfer Level(RTL). On the other hand, advanced algorithms which is used in various applications are more complicated. As a result, it needs much time to develop applications to be implemented in hardware than before.

In contrast, High-Level Synthesis(HLS) enables us to generate RTL from C, C++ and System-C code directly without hand code. In addition, HLS enable us to generate VHDL or Verilog simulation and test bench, as a result simulation and verification can be performed automatically.

To customize implementation, it is possible to set in-line expansion and configure memory access, interface etc, for optimization by using preprocessor called “Directive” in C-code which is used in HLS.

IV. Optimization method of Hardware implementation

We will show types of optimization method in this section. These optimization methods are speed up, implementation area, and target clock frequency. In addition, we use these methods in our experiment which will be described in Section V.

A. Speed up

Memory access is a well known bottleneck for speed up. Thus it is necessary for memory access to be more efficient.

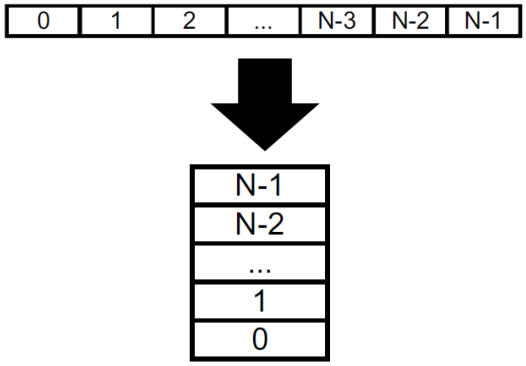


Fig. 4. A array partition[17]

First, we describe optimization of array access. The access of Block RAM in the Artix-7 FPGA has up to 2 I/O ports. Consequently, large number of access is necessary and the latency can be increased simultaneous access to large arrays. We can solve this problem with using the directive “ARRAY PARTITION”. This directive enables us to partition large arrays into multiple smaller arrays as shown in Fig. 4 to reduce the number of RAM access which became a bottleneck.

These arrays are not implemented in RAMs but in CLB as registers, so the implementation area is increased.

Second, we refer to block RAM and distributed RAM. We can select either block or distributed RAM by using directive “RESOURCE” in HLS. Block RAM is implemented on the Artix-7 FPGA, on the other hand, distributed RAM is implemented by using LUT in CLB. We use block RAM when reduction of CLB utilization is needed. However, the latency is increased when multiple access to a block RAM at the same time occurs. Hence, we use distributed RAM to decrease the latency because distributed RAM enables us to configure near the each logical elements which needs memory, and to access simultaneously. Contrary, CLB utilization is increased by using distributed RAM.

Finally, considering unroll the for-loop and in-line expansion of each function in C-code before High-level synthesis. In VIVADO HLS, it is possible to use directive “UNROLL” and “INLINE” for speedup. In default setting of HLS, all For-Loop is not expanded automatically and it causes to increase latency with divided process. This problem can be solved with “UNROLL” before for-loop in C-code. In addition, the latency is also increased with many functions because function hierarchy prevents reduction of latency and function call overhead. The directive “INLINE” removes function hierarchy and optimize these problems.

We optimize processing speed by mainly using 3 methods as mentioned above.

B. Implementation area

For optimizing implementation area, we consider memory and I/O access.

First, we consider memory access as follows. In AES processing, 10 times of memory access for loading S-box is needed in the case of processing full-round. If there is no limitation that all process of full-round AES completed in a clock, it is possible to reduce implementation area by implementing S-box in a block RAM. Contrary, in the case of optimizing speed, implementing S-box in distributed RAM or plural block RAM is needed. Owing to this rule, implementation area should be increased.

Second, we describe optimization I/O access. Usually, arrays of argument is partitioned into single array and loaded to some CLB registers for speed-up. However, implementation area is also increased. From this reason, we do not partition array and fulfill reduction of implementation area in exchange for speed-down.

We mentioned optimize implementation area by mainly using 2 methods as above.

C. Clock frequency

Before describing optimization, we will discuss the necessity of matching operation frequency of FPGA with its CPU. In FPGA implementation, long clock period enables the delay length of logical circuit to be long, and processing to be more efficient. Therefore, the lower clock frequency, the better latency is. For example, operation of FPGA in 1MHz is more efficient than in 100MHz.

On the other hand, Clock frequency is necessary to be high when we consider an entire hardware system. 1MHz operation of FPGA itself becomes the bottleneck in the hardware system no matter how its operation is fast enough. For this reason, it is necessary for operation frequency of FPGA to be near frequency of CPU for optimization.

Next, we consider target frequency. We pay attention to ARM Cortex-M3 which is used in various embedded systems such as car, factory, etc. Cortex-M3 is used in many evaluation board, and its frequency reaches almost 100MHz. For this reason, we decide that targeting frequency for optimize should be 100MHz.

Finally, we describe optimization of clock frequency. Typical optimization method is to divide a critical pass which is the longest pass in FPGA . It is possible to divide the critical pass by reducing condition divergence. An example of the condition divergence includes the multiplication over GF(2^8) in mixcolumn in AES. We set the period limit of critical pass as 10ns, and division of a critical pass can be performed automatically in HLS.

V. Experiment

In our experiment, we used each optimizing method of speed up, implementation area and target clock, moreover we did not use each method individually. We implemented 4 algorithms with these optimizing methods, so that, we implemented 12 patterns in our experiment.

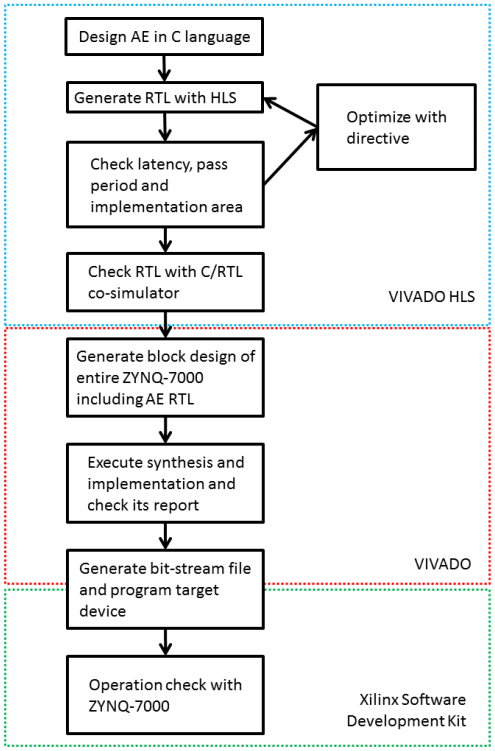


Fig. 5. The hardware implementation flow with HLS

A. The setting

In our hardware implementation, we used ZYNQ-7000 XC7Z020 and implemented with HLS. We consider implementation in embedded system, and set the message length as 16 Bytes. Supposing authenticated encryption for large data encryption, it is not necessary to use hardware circuits. There is no problem to use CPU with implemented accelerated circuit like a AES-NI, or GPGPU[5]. Altogether we think it has no benefit for hardware implementation of authenticated encryption with large data. By contrast, we think it is necessary that hardware implementation for embedded system which needs power saving, low cost and processing short message. Since we set message length as 16 Bytes.

For hardware implementation, we used Xilinx HLS for High-Level Synthesis and Xilinx VIVADO 2015.2 for implementation target. All strategy for high-level synthesis in HLS and implementation in VIVADO is set to default parameters. The following and Fig. 5 are our implementation flow.

1) Design authenticated encryption in C-code.
2) Generate RTL with HLS from C-code design and check latency and implementation area.
3) Optimize with directive.
4) Check RTL with C/RTL co-simulator.
5) Generate block design of entire ZYNQ-7000 evaluation board including authenticated encryption RTL.

[5]GPGPU:General-Purpose computing on Graphics Processing Units.

TABLE IV
IMPLEMENTATION RESULT OF AES-OTR[NB]

Optimization	Throughput [Mbps]	Implementation area[slices]	Clock [MHz]
Speed	278	2809	67.7
Area	122	2357	67.7
Clock	215	3263	100.7

TABLE V
IMPLEMENTATION RESULT OF SILC[NB]

Optimization	Throughput [Mbps]	Implementation area [slices]	Clock [MHz]
Speed	210	2328	82.4
Area	126	1986	82.4
Clock	147	3284	114.0

TABLE VI
IMPLEMENTATION RESULT OF AES-COPA[NR]

Optimization	Throughput [Mbps]	Implementation area [slices]	Clock [MHz]
Speed	228	2791	67.7
Area	117	2507	67.7
Clock	199	3509	105.6

6) Execute synthesis and implementation and check implementation area from implementation report.
7) Generate bit-stream file and program target device.
8) Verify generated design with ZYNQ-7000.

B. Measurements

At first, We generate 12 RTLs each of 4 algorithms and 3 optimization method with HLS. After synthesis, we confirm synthesis reports for all RLTs. The synthesis report contains summary of timing, latency, utilization and interface. Utilization is changed and improved by optimizing implementation process in VIVADO. On the other hand, theoretical value of timing and latency are hardly improved because the period of critical pass is almost fixed in HLS. That is why, we calculate throughput by using the value which reported in HLS. The following equation. 1 is calculation-formula for throughput.

$$Throughput[bit/sec] = \frac{Plaintext[Byte] \times 8 \times Targetfrequency[Hz]}{Latency(clockcycles)[times]} \quad (1)$$

Next, we generate block design of entire system of ZYNQ-7000 including authenticated encryption RTL. Then, we execute synthesis and implementation using the block design in VIVADO. After implementation, we open utilization report and confirm utilization of slices.

Finally, we program bit-stream file to FPGA, and confirm the operation to work normally by Xilinx Software Development Kit(SDK). The procedure of the confirmations are the following 3 steps.

1) Program bit-stream file to FPGA.
2) Make C-code program for executing encryption on FPGA.
3) Execute the program on CPU[6] and confirm that encrypt operation is normal.

Furthermore, the function for using FPGA is prepared by VIVADO and VIVADO HLS.

C. Analysis

We summarize the consequence of implementation for each algorithm in Table. IV, V, VI and VII. For these tables, optimization results are almost same as we expected.

However, area optimization of SILC is failed by the way which we described in Section IV-B. We show that consequence in Table. VIII. Area for speed optimization became larger than for area optimization after HLS. Although, area for speed up is smaller than for area after VIVADO implementation. We assume that not partitioning cipher text caused implementation area to large because SILC reuses cipher text for generating tag. Thereby we partitioned cipher text array in area optimization, and succeed to optimize for area(Table. V).

[6]CPU is Cortex-A9 in ZYNQ-7000 SOC

VI. CONCLUSION

In this paper, we described the optimization of implementation in FPGA with HLS. Our optimization with directives in HLS is confirmed to be effective. However, in optimization for area of SILC, it is not effective to use our optimization method as it is. Therefore, we should pay attention to the use of I/O data, and select either partitioning or not for each I/O data array. In consequence, optimization for area is succeeded. Based on this consequence, we will try to evaluate 4 authenticated encryptions in hardware implementation.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number 15K00113.

REFERENCES

[1] CAESAR committee, *CAESAR call for submissions, final*, http://competitions.cr.yp.to/caesar-call.html, 2014.
[2] Abolfazl Soltania and Saeed Sharifianb, *An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA*, Microprocessors and Microsystems Volume 39, Amirkabir University of Technology, Tehran, Iran, 2015.
[3] Semico Research *How an FPGA Approach to Complex System Design Can Improve Profitability: Real Case Studies* , http://www.xilinx.com/publications/prod_mktg/easypath-7-fpga-asic-approach.pdf, 2012.
[4] Xuejie Zhanga and Kam W Ng *A review of high-level synthesis for dynamically reconfigurable FPGAs*, Microprocessors and Microsystems Volume 24, Issue 4, The Chinese University of Hong Kong, China, 2000.
[5] Tom Feist, *White Paper: Vivado Design Suite, ver1.1*, XILINX, http://www.xilinx.com/support/documentation/white_papers/wp416-Vivado-Design-Suite.pdf, 2012.
[6] Farzaneh Abed, Christian Forler and Stefan Lucks, *Classification of the CAESAR Candidates*, Cryptology ePrint Archive:Report 2014/792. http://eprint.iacr.org/2014/792, 2014.

TABLE VII
IMPLEMENTATION RESULT OF POET[NR]

Optimization	Throughput [Mbps]	Implementation area [slices]	Clock [MHz]
Speed	251	3500	74.6
Area	124	3223	72.2
Clock	190	4533	100.7

TABLE VIII
AREA AFTER HLS VS AFTER VIVADO IMPLEMENTATION OF SILC

	Optimization for	LUT	Flipflop
After HLS	Speed	22408	4375
	Area	19499	2705
After VIVADO implementation	Speed	7949	7949
	Area	8212	8212

[7] Andrey Bogdanov, Martin M. Lauridse, and Elmar Tischhauser, *AES-Based Authenticated Encryption Modes in Parallel High-Performance Software*, Cryptology ePrint Archive:Report 2014/186, http://eprint.iacr.org/2014/186, 2014.

[8] *Intel 64 and IA-32 Architectures Software Developer's Manual*, http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf, 2015.

[9] Shay Gueron, *White Paper: Intel 繧托 Advanced Encryption Standard (AES) New Instructions Set*, Intel, https://software.intel.com/sites/default/files/article/165683/aes-wp-2012-09-22-v01.pdf, 2012.

[10] Kazuhiko Minematsu.:*AES-OTR v2*, http://competitions.cr.yp.to/round2/aesotrv2.pdf (2015).

[11] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, Sumio Morioka, and Eita Kobayashi.: *SILC v2*,http://competitions.cr.yp.to/round2/silcv2.pdf (2015).

[12] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda.:*AES-COPA v.2*, http://competitions.cr.yp.to/round2/aescopav2.pdf (2015).

[13] Farzaneh Abed, Scott Fluhrer, John Foley, Christian Forler, Eik List, Stefan Lucks, David McGrew, and Jakob Wenzel.: *The POET Family of On-Line Authenticated Encryption Schemes v2.0*, http://competitions.cr.yp.to/round2/poetv20.pdf (2015).

[14] *zynq7000 product table*, XILINX, http://www.xilinx.com/support/documentation/selection-guides/zynq7000-product-table.pdf, 2015

[15] XILINX, *Technical Reference Manual: Zynq-7000 All Programmable Soc, ver 1.10*, http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf, 2015.

[16] *User Guide: 7 Series FPGAs Configurable Logic Block*, XILINX, http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf, 2014.

[17] XILINX, *Vivado Design Suite User Guide: High-Level Synthesis, ver 2015.2*, http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug902-vivado-high-level-synthesis.pdf, 2015.