Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186–5140
Volume 4, Number 1, pages 1–8, January 2015

FPGA implementation of Ciphers using Schematic to Program Translator(SPT)

Masashi Watanabe, Keisuke Iwai, Hidema Tanaka, and Takakazu Kurokawa
National Defense Academy of Japan
Kanagawa, Japan
Email:{em52037,iwai,hidema,kuro}@nda.ac.jp

***Abstract*—With the spread of heterogeneous computing, accelerators such as GPU are widely used. However, it is not easy to develop a software program that runs at high speed on accelerators. On the other hand, encryption algorithms are evaluated with not only the strength but also the implementability and the performance. Therefore it is important to compare their performance by throughput using accelerators. We proposed a development tool named SPT(Schematic to Program Translator) for high-speed processing of encryption as well as GPU and FPGA. In this paper, we discussed FPGA implementation of cipher using SPT. In this tool, a C program is automatically generated from figures drawn in accordance with the specifications of the encryption algorithm. These programs are adjusted for the C compiler, CUDA translator and high-level synthesis tool. Moreover, many-core processors, GPU and FPGA can be easily used by passing these programs to the C compiler, CUDA translator and high-level synthesis tool. As a result, circuits generated by high-level synthesis tool(VivadoHLS provided by Xilinx) using C programs generated by SPT can perform encryption process correctly on FPGA, and their performance became faster than manually generated codes.**

***Keywords*—GUI, Implementation of Encryption Circuit, High-Level Synthesis, FPGA, SoC, AES, Camellia**

I. Introduction

The study on high speed calculation with good energy efficiency is performed by various approaches. A many-core processor represented by GPGPU which used GPU for general-purpose processing [1] [2][3][4], the distributed processing using general-purpose processors and reconfigurable computing using FPGA are paid attention. The spread of heterogeneous computing that has more than two kinds of processers are equipped by many commercially available devices such as computer or smartphone. Programs or circuits are implemented for GPU using CUDA or OpenCL etc and for FPGA using VerilogHDL, VHDL or High-Level Synthesis etc. However, it is not easy to develop a software program that runs at high speed on accelerators such as GPU or FPGA etc using these languages or tools.

On the other hand, encryption algorithms are evaluated with not only the strength but also the implementability and the performance[5]. Therefore it is important to compare their performance by throughput using accelerators. Accordingly, encryption algorithms developers have to take account of performance comparison using the latest accelerators.

Although reconfigurable computing achieves high power efficiency, the design of a circuit on FPGA takes time and effort than software. As a mean to solve these problems, high-level synthesis is paid attention. Although it is mainstream to describe the design of a circuit to implement on FPGA using hardware description languages (HDL) such as VHDL or Verilog HDL in register-transfer-level (RTL), high-level synthesis tool provides easier design environment than HDLs and various automatic optimization.

Thereby we proposed a development tool named SPT(Schematic to Program Translator) to make it easy to use high-speed processing of encryption as well as FPGA and many-core processor. In this tool, a C program is automatically generated from figures drawn in accordance with the specification of the encryption algorithm. In addition, these programs are implemented on CPU as well as GPU and evaluated [6][7]. In this paper, we will discuss FPGA implementation of cipher using SPT. We also show our implementation results of two kinds of encryption circuits (AES[8] and Camellia[9]) on FPGA using Vivado HLS which is a high-level synthesis tool provided by Xilinx, and compare them on the standpoint of speed, area and speed per area.

The remainder of this paper is organized as follows. In the next section, we indicate conventional problems of implementation of symmetric block cipher on accelerators and brief explanation of SPT. In section III, IV and V, constitution, design and implementation of SPT are shown. Code generation for high-level synthesis tool(Vivado HLS) by SPT, its evaluation results are explained in section VI. Finally, we explain our conclusion in section VII.

II. Implementation of symmetric block cipher on accelerators

A. Conventional implementation methods

The symmetric block ciphers can be realize by software implementation as well as hardware implementation. In software implementation, programming languages and parallel processing libraries are used. Similarly, the schematic drawing, hardware description languages(HDL) and high-level synthesis(HLS) etc are used for hardware implementation.

B. Conventional problems of implementation

For software implementation, programming languages such as the C language are used for CPUs. CUDA and OpenCL which are extended from these languages are used for GPUs. For hardware implementation, hardware description languages(HDL) or high-level synthesis tools such as SystemC

are used for FPGA. Both implementation require enough knowledge and skills to describe programs adequate to each accelerator. It is not easy to describe programs with high performance. Moreover when we have to implement the same algorithm to a different device, it is necessary to renew programs. However, this renewal requires enough knowledge and skills with a waste of time and labor. Similarly they are not easy.

The study on software as well as hardware implementation of symmetric block ciphers using prepared template of known algorithms with GUI interface is performed in [10]. On the other hand, we proposed SPT in [6] which has high flexibility by drawing figures which accord with algorithms of symmetric block ciphers, and it can be applied to other fields than block ciphers and versatile availability by combining with a translator.

Drawing figures that had higher level of abstraction than programming descriptions such as C language are used for input of SPT. Because HLS makes it possible to implement circuits form the description that had high level of abstraction such as the C language to hardware.

C. Flow of the SPT processing

Figure 1 shows summary of SPT. The input of SPT is figures, while the output of SPT are the codes which can be adapted for C compiler, CUDA translator and high-level synthesis tool. Programs or circuits are implemented to CPU, GPU or FPGA. Figure 2 shows an example of drawing figures for input to SPT. A program to perform $y = a+b \oplus 0xffff$ is generated when this figure is input into SPT. In this figure, the quadrangular element (node) expresses arguments of functions, constant number or kinds of the processing and the arrow (arc) express data flow.

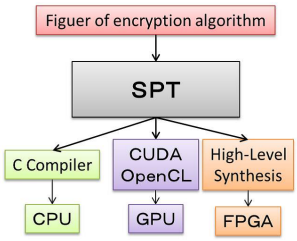

Fig. 1. Summary of SPT

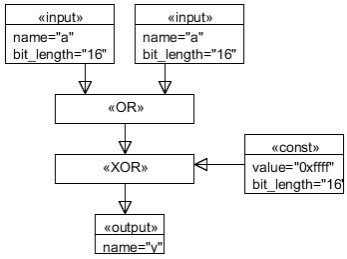

Fig. 2. Example of drawing figures

III. Constitution of SPT

Figure 3 shows constitution of SPT. It consists of four parts listed as follows:

(1) Figure input part
(2) Figure analysis part
(3) Data flow analysis part
(4) Code generate part

Three intermediate representation(IR) are used between these parts as follows:

(1) UXF(UML Exchange Format)
(2) SML(Schematic Markup Language)
(3) DML(Dataflow Markup Language)

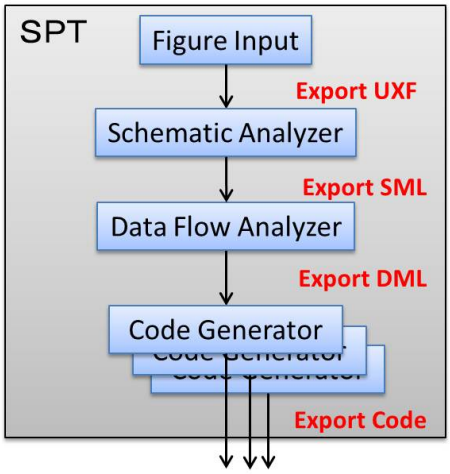

Fig. 3. Constitution of SPT

IV. Design of SPT

Three packages are shown in Fig.4. UMLet[11] and these packages are main components of SPT. UMLet is a front-end of SPT and corresponds to the figure input part. UXFtoSML package corresponds to the figure analysis part. SMLtoDML package corresponds to the data flow analysis part. CodeGenerator package corresponds to the code generate part. Each package has element package and interpreter package, which are class groups to express elements to output and to parse IR.

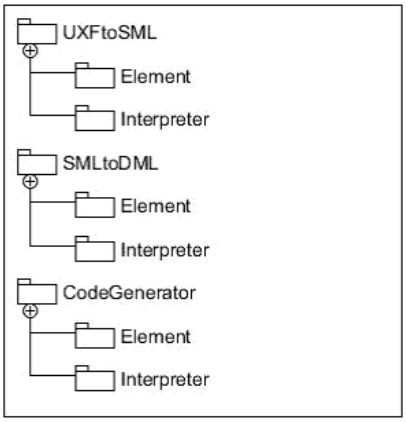

Fig. 4. Three packages constituting SPT

A. UXFtoSML package

In SPT, a quadrangle is treated as a node which expresses a kind of the processing, and an arrow is treated as an arc which expresses the data flow. However, UXF has only form and position of elements because it is a format to express a

figure of UML. Therefore the kind of processing is decided by a keyword and parameters described in a quadrangle. This package analyzes the form, the keyword and parameters of element. Moreover a quadrangle is converted into a node that expresses a kind of processing. Similarly an allow is converted into an arc that expresses the data flow. These conversance results became output of schematic analyzer in the form of SML.

B. SMLtoDML package

In this package, relations of the node and the arc are converted into a task graph from the positional information of each element in SML. This task graph is a directed acyclic graph(DAG), and topological sort is applicable to this graph. For this reason, scheduling is carried out using topological sort, and the set of an order taking a sequential step is generated as output in the form of DML.

C. CodeGenerator package

The code of C program for C compiler, CUDA translator and high-level synthesis tool is generated in this package. SPT has already been able to generate the code for C compiler and CUDA translator [6][7]. In this study, we added another ability to SPT to be able to generate the code for high-level synthesis tool.

V. Implementation of SPT

SPT was developed by Java. Because we selected an object-oriented language which is suitable for making the code easier to change or to add a new feature as the programming language to implement SPT. Additionally, Java can be used without depending on the OS.

A. Operating method

The operating method of UMLet which is the front-end of SPT is shown in this section. Figure 5 shows input screen of UMLet. In this screen, the part surrounded in a red frame is named a diagram panel, the part surrounded in a blue frame is named a palette panel and the part surrounded in a green frame is named a property panel. Elements in palette panel are copied in diagram panel by double clicking or drag-and-dropping. The parameter of each element can be edited in property panel. The figure in conformity with specifications of block cipher is completed by connecting each element with arrows.

B. Drawing figures

Figures 6-9 show the specification figures and the drawn figures as examples to implement Camellia. The left side of each figure shows a specifications figure, while the right side shows a figure drawn using UMLet. Camellia's encryption function, F function, FL function, and FL^{-1} function are shown in Fig.6, Fig.7,Fig.8 and Fig.9 respectively.

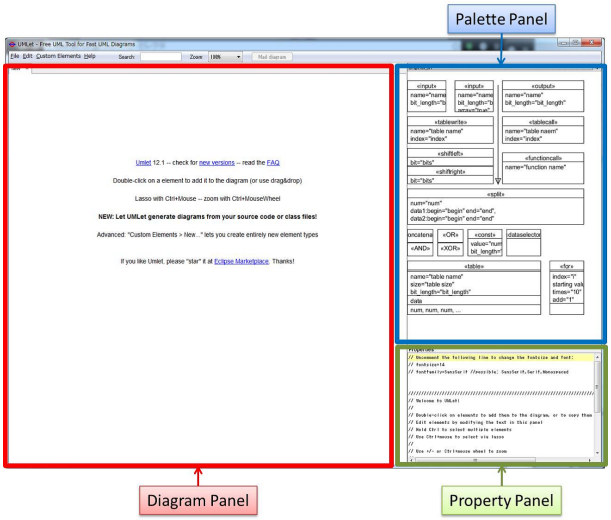

Fig. 5. Input screen of UMLet

VI. Code generation for high-level synthesis tool(VivadoHLS)

Various features such as optimization of array, loop unrolling and function inlining are available using directives in high-level synthesis tool(VivadoHLS). The arrarys including substitution tables or extended keys can be accessed faster by optimization using directives or preparation of copies. This fact was confirmed in out previous study[12]. In consideration of these findings, the code for high-level synthesis tool is generated in SPT as follows.

A. Declaration of the variable

The arbitrary precision integer data types are available in C program for VivadoHLS. Therefore all variables are declared using the arbitrary precision integer data types in the code which is generated by SPT. For example, a declaration

```
uint16 n
```

accords with a variable called n with 16 bits length.

B. Parallel access to array

Arrays in which extended keys or substitution tables are stored are implemented on registers or distributed RAM. As a result, they can be accessed in parallel. The details are shown below.

1) Parallel access to extended keys: Extend keys are implemented on registers by using directives because they are not so large size such as 10 keys which have 128-bit length in AES and 22 keys which have 128-bit length in Camellia. For an example, a directive described by

```
#pragma HLS ARRAY_PARTITION variable=exkey complete dim=1.
```

means that an extended key called as "exkey" is stored at a register.

2) Parallel access to substitution tables: Copies of substitution tables are prepared and are implemented on distributed RAM because their sizes are large. Therefor, (The number of times that is accessed / 2) substitution tables are

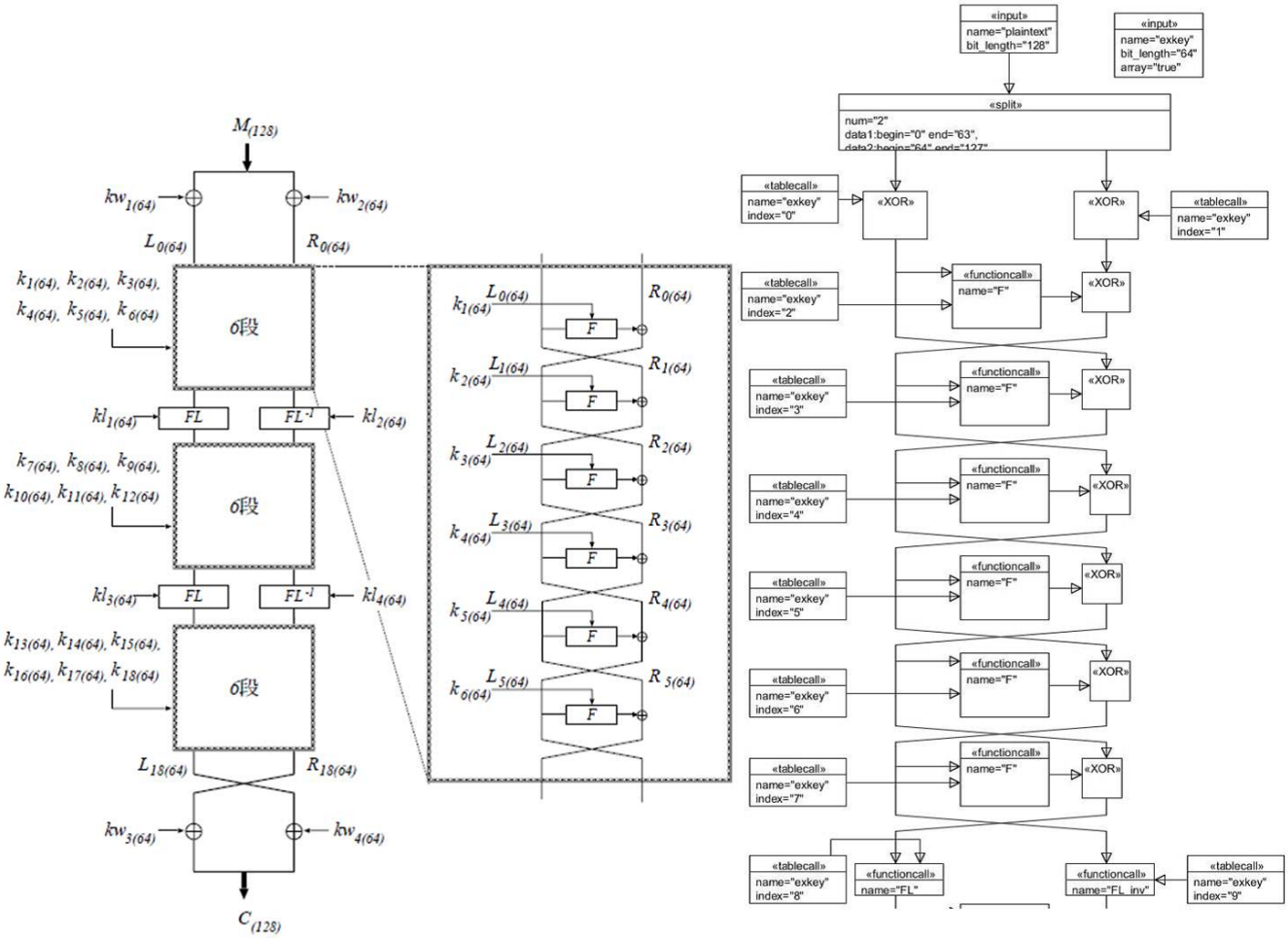

Fig. 6. Example of drawing figure(Camellia's encryption function)

generated in code generator of SPT by using dual port RAMs which can be accessed doubly at the same time. These tables are implemented on the distributed RAM by a directive as follows:

```
#pragma HLS RESOURCE variable=sp1_0
core=RAM_2P_1S.
```

C. Evaluation

We draw figures of AES and Camellia, and input them into SPT. The code generated from these figures by SPT is implemented on FPGA. Figure 10 shows a part of Camellia's code generated by SPT for VivadoHLS. The precise circuits were synthesized by VivadoHLS from these codes, and these circuits were confirmed to encrypt exactly. These circuits are evaluated by throughput, area and throughput par area. The evaluation environment is summarized in the Table I.

TABLE I. EVALUATION ENVIRONMENT

O S	Windows 7 Professional
CPU	Intel Core i7-920 3.4GHz
Memory	16GB
High-Level Synthesis Tool	VivadoHLS
FPGA	ZYNQ(xc7z020clg484-1)

D. FPGA implementation results

Table II shows FPGA implementation results. AES become about three times faster than Camellia in throughput. However, the circuit area of AES become larger than Camellia. As a result, performance of AES becmae almost the same as Camellia in Throughput par Area.

TABLE II. FPGA IMPLEMENTATION RESULTS

Algorithm	Throughput [Mbps]	Area [slices]	Throughput/Area [Kbps/ slices]
AES	1789.1	9158	200.0
Camellia	670.0	3403	201.6

E. Comparison with handmade

Table III shows FPGA implementation results of AES presented in our previous study [12]. Each type of implementation is shown as follows:

Type1 A substitution table (S-box) is implemented as a look-up table.

Type2 Sixteen S-boxes are implemented as a look-up table.

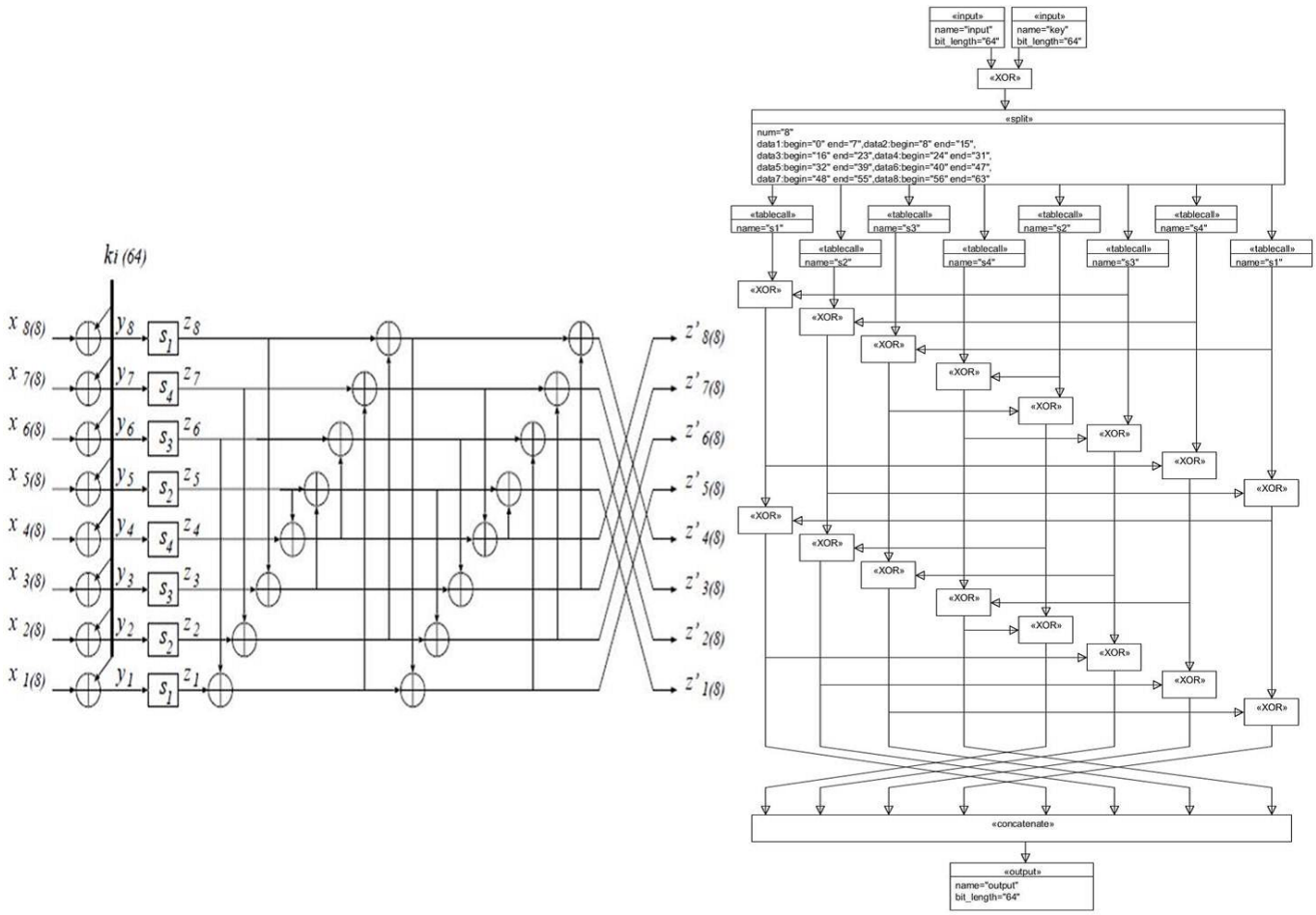

Fig. 7. Example of drawing figure(Camellia's F function)

Type3 S-box is implemented by calculation using inverse element in a Galois field and an affine transformation.

Type4 A substitution table (T-box) which combines a calculation result of MixColumns and S-box is implemented.

In addition, we described a code named Type5 extended from Type4, which applied loop unrolling. This type became as fast as TypeSPT.

TABLE III. FPGA IMPLEMENTATION RESULTS OF AES.

Type	Throughput [Mbps]	Area [slices]	Throughput/Area [Kbps/ slices]
Type1	96.7	455	217.6
Type2	448.1	403	1138.6
Type3	37.1	563	66.3
Type4	1393.5	646	2208.9
Type5	1789.1	4576	400.4
TypeSPT	1789.1	9158	200.0

The last row named as "TypeSPT" uses codes generated by SPT. The circuit generated by SPT using TypeSPT became faster than other former results.

Performance of circuits depended on the implementation types. However, implementation results show that the speed of the circuits which were high-level synthesized using codes described by hands depended on the ability of the person who describes the circuit. For this reason, enough speed may not be provided in the circuit which was high-level synthesized from the code that was described by hands, and the circuit generated by SPT could not reached very high-performance at all, but can achieve stable high-speed.

As a result, the code that was generated by SPT achieved the speed that was as fast as, or faster than the codes described by hands. This is attributed to static single assignment(SSA). Because SPT uses SSA form to generate codes, the circuit generated by SPT runs fast because of the high-level synthesis tool which uses registers effectively. However, its area increased. Thereby, Type5 became 2 times faster than TypeSPT in cases of AES(Throughput/Area).

VII. CONCLUSION

Two kinds of encryption circuit were implemented on FPGA using SPT that is the development tool to make it easy to use high-speed encryption processing.

Codes of AES and Camellia for high-level synthesis tool are generated by SPT. These circuits are synthesized by VivadoHLS which is a high-level synthesis tool provided by Xilinx from these codes. We compared their throughputs and

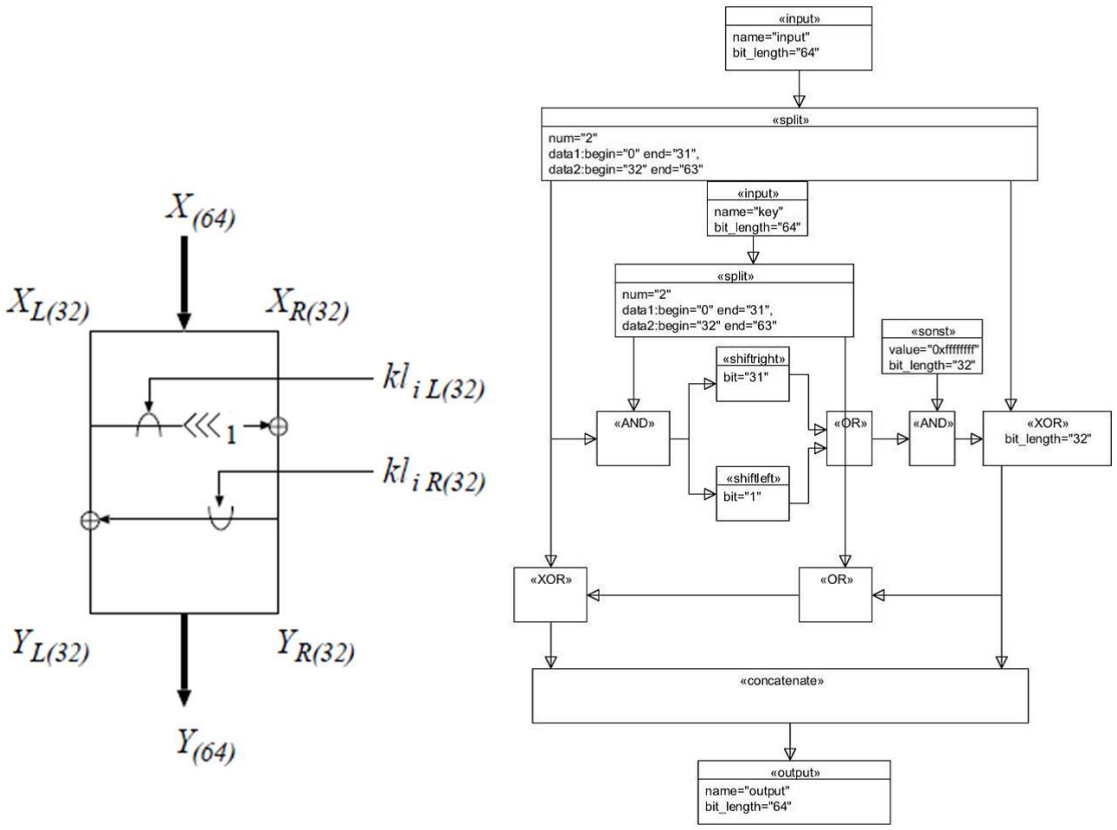

Fig. 8. Example of drawing figure(Camellia's FL function)

evaluated these circuit. As a result, the code that was generated by SPT achieved the speed that was as fast as, or faster than the codes that were described by hands. It can be thought that the circuit run fast because of the use of SSA. However, its area increased.

Finally, SPT is the development tool from figures which ware drawn in conformity with specifications of block ciphers. SPT can generate codes for CPU, GPU and HLS. The purpose of our study is to give high performance for software implementation using many core processor, SIMD and hardware implementation using HLS. Therefore, we will further improve the quality of SPT.

ACKNOWLEDGMENT

This work was supported by Japan Society for the Promotion of Science (JSPS) Grant-in-Aid for Scientific Research (KAKENHI) Grant Number 25871223. (Grant-in-Aid for Young Scientists B)

REFERENCES

[1] N. Nishikawa, K. Iwai, and T. Kurokawa, "High-performance symmetric block ciphers on cuda," in *Proceedings of the 2011 Second International Conference on Networking and Computing*, ser. ICNC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 221–227. [Online]. Available: http://dx.doi.org/10.1109/ICNC.2011.40

[2] D. A. Osvik, J. W. Bos, D. Stefan, and D. Canright, "Fast software aes encryption," in *Proceedings of the 17th international conference on Fast software encryption*, ser. FSE'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 75–93. [Online]. Available: http://dl.acm.org/citation.cfm?id=1876089.1876096

[3] K. Iwai, T. Kurokawa, and N. Nishikawa, "Aes encryption implementation on cuda gpu and its analysis." in *ICNC*. IEEE Computer Society, 2010, pp. 209–214. [Online]. Available: http://dblp.uni-trier.de/db/conf/ic-nc/ic-nc2010.html#IwaiKN10

[4] S. Manavski, "Cuda compatible gpu as an efficient hardware accelerator for aes cryptography," in *Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on*, Nov 2007, pp. 65–68.

[5] National Institute of Information and Communications Technology(NICT), Information-technology Promotion Agency Japan(IPA), *CRYPTREC Report 2012*, 2013. [Online]. Available: http://www.cryptrec.go.jp/report/c12_sch_web.pdf

[6] M. WATANABE, K. IWAI, H. TANAKA, and T. KUROKAWA, "Development of cipher implementation tool using gui in japanese," pp. 125–129, july 2014.

[7] ——, "Gpu implementation of ciphers using schematic to program translator(spt) in japanese," pp. 35–42, oct 2014.

[8] National Institute of Standerd and Technology(NIST), *FIPS-197 Advanced Encryption Standard(AES)*, 2001.

[9] Nippon Telegraph and Telephone Corporation, Mitsubishi Electric Corporation, *Specification of Camellia - a 128-bit Block Cipher*, 2001.

[10] A. Khalid, M. Hassan, A. Chattopadhyay, and G. Paul, "Rapid-feinspn: A rapid prototyping framework for feistel and spn-based block ciphers." in *ICISS*, ser. Lecture Notes in Computer Science, A. Bagchi and I. Ray, Eds., vol. 8303. Springer, 2013, pp. 169–190. [Online]. Available: http://dblp.uni-trier.de/db/conf/iciss/iciss2013.html#KhalidHCP13

[11] GNU, "Umlet 12.2," http://www.umlet.com/, 2013.

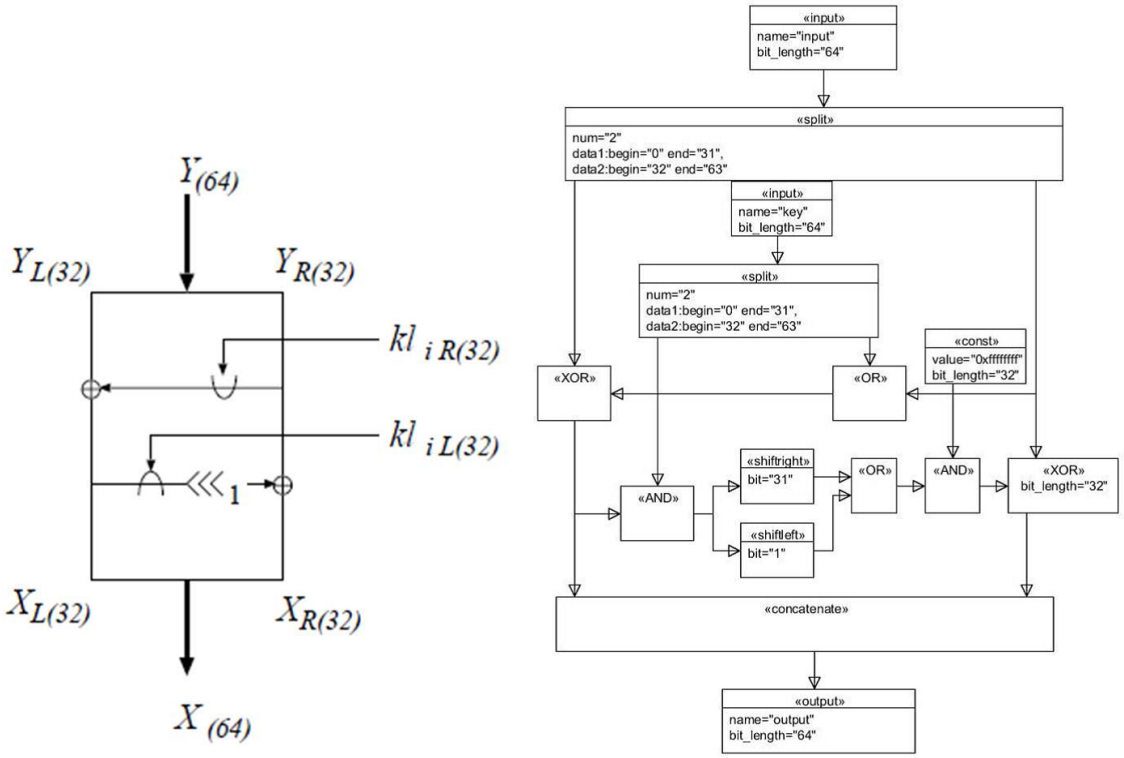

Fig. 9. Example of drawing figure(Camellia's FL^{-1} function)

[12] M. WATANABE, K. IWAI, H. TANAKA, and T. KUROKAWA, "High-speed implementation of encryption circuit using a high-level synthesis tool," pp. 63–66, Feb 2013.

```
#include "camellia_test.h"
uint128 camellia(uint128 plaintext, uint64 exkey[26]){
#pragma HLS ARRAY_PARTITION variable=exkey complete dim=1
#pragma HLS RESOURCE variable=plaintext core=AXI4LiteS metadata="-bus_bundle io"
#pragma HLS RESOURCE variable=return core=AXI4LiteS metadata="-bus_bundle io"
        uint64 sp1_0[256] = {···};
                            ·····
        uint64 sp7_8[256] = {···};
        #pragma HLS RESOURCE variable=sp7_8 core=RAM_2P_1S
        var_0_0 = exkey[0];
                            ·····
        var_0_70 = exkey[25];
        var_0_1 = (plaintext >> 64) & 0xffffffffffffffff;
        var_0_3 = plaintext & 0xffffffffffffffff;
        var_0_4 = var_0_0 ^ var_0_1;
        var_0_6 = var_0_3 ^ var_0_2;
        F_F_var_1_19 = var_0_4 ^ var_0_5;
        F_F_var_1_1 = (F_F_var_1_19 >> 56) & 0xff;
        F_F_var_1_2 = (F_F_var_1_19 >> 48) & 0xff;
        F_F_var_1_3 = (F_F_var_1_19 >> 40) & 0xff;
        F_F_var_1_4 = (F_F_var_1_19 >> 32) & 0xff;
        F_F_var_1_5 = (F_F_var_1_19 >> 24) & 0xff;
        F_F_var_1_6 = (F_F_var_1_19 >> 16) & 0xff;
        F_F_var_1_7 = (F_F_var_1_19 >> 8) & 0xff;
                            ·
                            ·
                            ·
        F_F_var_1_8 = F_F_var_1_19 & 0xff;
        F_F_var_1_9 = sp1_8[F_F_var_1_1];
        F_F_var_1_11 = sp2_8[F_F_var_1_2];
        F_F_var_1_13 = sp3_8[F_F_var_1_3];
        F_F_var_1_15 = sp4_8[F_F_var_1_4];
        F_F_var_1_16 = sp5_8[F_F_var_1_5];
        F_F_var_1_10 = sp6_8[F_F_var_1_6];
        F_F_var_1_12 = sp7_8[F_F_var_1_7];
        F_F_var_1_14 = sp8_8[F_F_var_1_8];
        F_F_output = ((((((F_F_var_1_9 ^ F_F_var_1_11) ^ F_F_var_1_13) ^ F_F_var_1_15)
                                ^ F_F_var_1_16) ^ F_F_var_1_10) ^ F_F_var_1_12) ^ F_F_var_1_14;
        var_0_63 = F_F_output;
        var_0_67 = var_0_69 ^ var_0_65;
        var_0_66 = var_0_63 ^ var_0_62;
        var_0_68 = var_0_66 ^ var_0_70;
        ciphertext = ((uint128)var_0_67 << 64) ^ var_0_68;
        return ciphertext;
}
```

Fig. 10. A part of Camellia's code generated by SPT