COMPACT FPGA IMPLEMENTATION OF CAMELLIA

Panasayya Yalla and Jens-Peter Kaps

Volgenau School of IT&E, George Mason University
Fairfax, VA, USA
email: {pyalla, jkaps}@gmu.edu

ABSTRACT

We present the smallest FPGA implementation of Camellia for 128-bit key length to date. This architecture was designed for low area and low power applications. Through specific optimizations such as shift registers for storing and scheduling key, distributed RAM for storing data, we achieved compact implementation using only 318 slices at a throughput of 18.41Mbps on the smallest Xilinx Spartan-3 XC3S50-5 device.

1. INTRODUCTION

The Camellia algorithm [1] was jointly developed by Nippon Telegraph and Telephone Corporation (NTT) and Mitsubishi in 2000. It was designed for a wide range of design platforms from low power and limited resources to high performance on multiple platforms. However, the main design goal was security. The New European Schemes for Signatures, Integrity, and Encryption (NESSIE) project has nominated Camellia as a strong block cipher in 2003 [2]. The structure of Camellia provides features for a compact design. Several different attacks were performed successfully only on reduced round versions of Camellia. An impossible differential cryptanalysis on reduced round Camellia is described in [3], collision attacks in [4, 5]. The best know attack can break 9-rounds of Camellia with 128-bit key [4].

2. CAMELLIA

Camellia [1] is a 128-bit block cipher which supports key lengths of 128, 192 or 256 bits. In this paper, we describe an implementation with 128-bit key length. The Camellia algorithm uses a Feistel network with pre-whitening before first and post-whitening after last rounds. The functions, FL and FL^{-1} are inserted after 6^{th} and 12^{th} round introduce non-regularity. The block diagram of 128-bit encryption

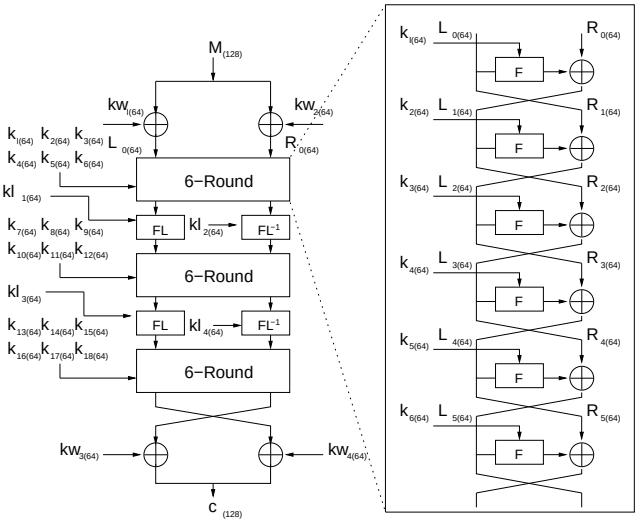

Fig. 1. Block diagram of 128-bit key encryption

can be seen in Fig1. The F-function contains a Substitution-Permutation Network (SPN) which is composed of non-linear S-function and linear P-function. The S-function consists of 8 S-Boxes which are selected from four different types. The P-function is comprised of byte permutations. The block diagram of the F-function is shown in Fig 2. The key schedule generates round keys of 64-bit size by shifting the original key K_L and the modified key K_A. Computation of K_A is described in the Section 2.3.

2.1. Notations

X_L	left-half data of X.
X_R	right-half data of X.
$\oplus$	bitwise exclusive-OR operation.
$\|\|$	concatenation of two operands.
$\lll_n$	circular rotation to left by n bits
$\ggg_n$	circular rotation to right by n bits
$\cup$	bitwise AND operation
$\cap$	bitwise OR operation

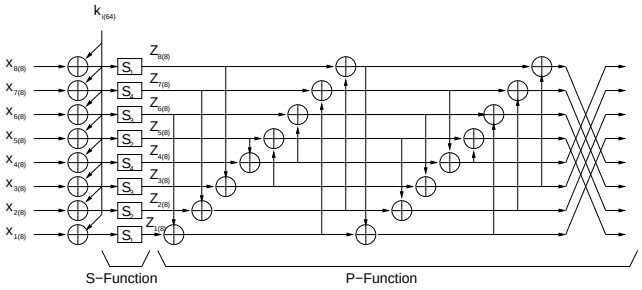

Fig. 2. F-function

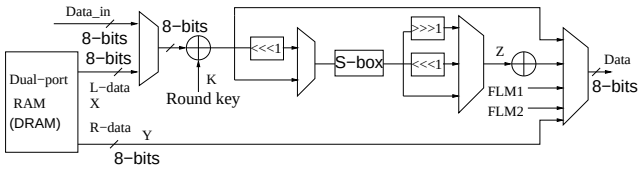

Fig. 3. Top level Block Digram

2.2. Encryption for 128-bit key

The 128-bit plain text M_{128} is XORed with pre-whitening key $kw_1 \| kw_2$ and separated into two halves L_0 and R_0 each of 64-bit size. L_0 is then passes through the F-function where it is XORed with round key k_0 The result is applied to the S-Boxes and output of S-box to P-function which is XORed with right half of the data R_0. At the end of each routine the left half and the right half of the data are swapped. The same process is repeated for all the 18 rounds. The 6^{th} and 12^{th} rounds, the left half of the data $L_r^{'}$ is given to FL and right half $R_r^{'}$ to FL^{-1}.

2.3. Key schedule

In the first phase of the key schedule, the modified key K_A is computed from the original key K_L. In the second phase, round keys are generated through rotation of K_L or K_A by 15 or 17-bits.K_A is computed by passing K_L through 4 rounds of the same Feistel network which is used for encryption with XOR of K_L after the 2^{nd} round. The round keys used are four constant

3. COMPACT ARCHITECTURE OF CAMELLIA

Our goal is to design a very compact architecture for small area with an acceptable throughput. We choose to implement our architecture on Xilinx spartan-3 family FPGA devices. Our architecture uses a 8-bit datapath and does both encryption and key scheduling. Figure 3 shows the top level block diagram of our architecture. We tried different implementation strategies for several component used in the architecture to get the best results.

3.1. S-Boxes and F-Function

The S-Boxes S_2, S_3, S_4 can be derived from S-Box S_1 as S_2(x)=S_1(x)$\lll_1$, $S_3(x)$=$S_1(x) \ggg_1$, $S_4(x)$=$S_1(x \lll_1)$. This can be realized in hardware through one S-Box and two multiplexers. Hence instead of 8 S-Boxes, only one S-box is required which reduces the area by 85% .

Each Xilinx Spartan 3 Configurable Logic Block (CLB) contains two SLICEMs which in turn contain two Look-Up Tables (LUTs). The LUTs can be configured as a Distributed RAM (DRAM) or a 16-Bit shift register (SRL16) apart from implementing logic. The DRAMs are used for fast and efficient implementation of memory. In this architecture, a dual port 16x8 Distributed RAM (DRAM) is used for storing the data which reduces the area by approximately 75% compared to using a 128-bit register.

In this implementation, the 64-bit F-function is broken down into several of 8-bit operations. The XOR of 8-bits of the left data X and 8-bits round key K passes through S-Box. The result is XORed with of corresponding 8-bits of right data Y. This is repeated depending on the number of XORs required to complete the P-function. XORing the output from S-box with right data saves storage required for the intermediate values. For this reason a dual port DRAM is used. The swapping of data is accomplished by addressing. It takes 44 clock cycles to complete one round. The multiplexers before the 1^{st} XOR and after the 2^{nd} XOR operation enable the computation of the modified key from the original key and the pre-and post whitening operation.

3.2. FL and FL^{-1}

The FL function breaks its 64-bit input into two 32-bit halves namely X_L and $X_R$and similarly 64-bit key kl as kl_L and kl_R. The FL 'is broken into two parts FL_1 and FL_2 and FL^{-1} function into FL_1^{-1} and FL_2^{-1}.

$$FL_1(X_L, X_R, kl_L) = ((X_L \cap kl_L) \ggg_1) \oplus X_R \quad (1)$$
$$FL_2(X_L, X_R, kl_R) = (X_R \cup kl_R) \oplus X_L \quad (2)$$
$$FL_1^{-1}(X_L, X_R, kl_R) = (X_R \cup kl_R) \oplus X_L \quad (3)$$
$$FL_2^{-1}(X_L, X_R, kl_L) = ((X_L \cap kl_L) \lll_1) \oplus X_R \quad (4)$$

As can be seen from equations 1 and 4, FL_1 and FL_2^{-1} are the same operation and Similarly FL_2 and FL_1^{-1} from equations 2 and 3. Hence, we combine FL_1 and FL_2^{-1} as one function called FLM_1 and FL_2 and FL_1^{-1} as FLM_2.

$$FLM_1(X_L, X_R, kl_L) = ((X_L \cap kl_L) \lll_1) \oplus X_R \quad (5)$$
$$FLM_2(X_L, X_R, kl_R) = (X_R \cup kl_R) \oplus X_L \quad (6)$$

This saves two XORs needed for FL and FL^{-1} operations. The 32-bit cyclic rotation in FLM_1 is implemented as a 1-bit left shift on 8-bit data with one flipflop to store the shifted bit. After completing FLM_1, the last bit is computed again to get the correct bit.

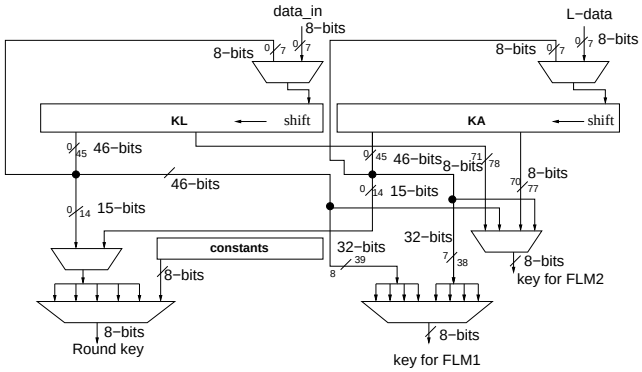

Fig. 4. Key scheduling using shift registers.

3.3. Key Storage and Scheduling

The Camellia algorithm needs two keys of size 128 bit, the original key K_L and the modified key K_A, which are rotated to generate the round keys. An efficient way of storing the keys is to either using SLICEM LUTs as a DRAM or using as shift register. Addressing for DRAM is complicated as key scheduling involves shifting. The best choice is to store the keys in a LUT based shift register. However, such a shift register has only a single bit output and each output or tab requires a flip-flop. Hence, the area consumed by such a shift register depends mainly on the number of taps required to access the data. All the shift registers in this implementation shift by 8-bit in order to match the width of the datapath. However, the rotations needed for round key generation are 15, 30, 45, 60, 77, 94 and 111-bits. as shown in Table 1. We can accomplish this by 8-bit shifts and an 8-bit 5:1 multiplexer as $n \bmod 8$ has only 5 different results. In order to make the control logic simple and uniform, shifting of the key is done at the last clock cycle of the round. For normal round key generation, tapping 15-bits is sufficient. However, due to FLM_1 which has a 32-rotation, 41-bits additional tabs are required. This increased the size of the shift register approximately by 2 folds. The key scheduling can be seen in Figure 4. The original key K_L is initially loaded into both DRAM and K_L shift register. The constants for generating modified key K_A are stored in a seperate shift register. K_A is computed using the datapath from Fig 3. It is loaded into the K_A shift register, while data is loaded into DRAM. Using shift registers for both K_L and K_A reduces the area by about 75% compared to using two 128-bit registers.

3.4. Controller

The control unit consists of a main controller and sub controllers for the F- and FLM functions. The main controller stores its control words in as Distributed ROM (DROM) for the reasons stated in Section 3.1. The address for the control word is generated by a 6-bit counter. Using a DROM and a counter for the main controller, its size is reduced by 97% as compared to a FSM. The sub controllers stores their control words in a shift registers as they repeat a sequence of operations.

Table 1. Number of Shifts of Shift register

Round keys	Rotation of key (n)	Number of 8-bit shifts	Relative shifts	$n \bmod 8$
kw_1,kw_2,k_1,k_2	0	0	0	0
k_3,k_4,k_5,k_6	15	1	1	7
kl_1,kl_2	30	3	2	6
k_7,k_8,k_9	45	5	2	5
k_{10},k_{11},k_{12}	60	7	2	4
kl_3,kl_4	77	9	2	5
k_{13},k_{14},k_{15},k_{16}	94	11	2	6
kw_3,kw_4,k_{17},k_{18}	111	13	2	7

Table 2. Components for Camellia Implementation

Component	Implementation	slices
F-controller-(1)	FSM	40
F-controller-(2)	Shift Register	2
Key storage-(1)	Register	128
Key Storage-(2)	Shift Register	32
Controller-(1)	FSM	214
Controller-(2)	DROM	40
Data storage-(1)	Register	64
Data storage-(2)	DRAM	16

4. RESULTS

We implemented our design on the smallest device of the Xilinx Spartan-3 FPGA family using Xilinx ISE 9.2i for synthesis and Active HDL 7.2 for simulation. The results are verified with the test vector provided in [2]. Table 3 shows the results of different implementations strategies of the major components of camellia. Using the components denoted by (1), the total area of the design is 800 slices due to huge multiplexers used for key which are not shown in Table 3. Using components denoted by (2), the total area is reduced to 318 slices which is the smallest Camellia implementation to date. We implemented two versions of Camellia namely Camellia-2a and Camellia-2b using implementation strategy (2). Camellia-2a uses LUT based S-Box and Camellia-2b uses Block RAM. The results of these implementations are shown in Table 3 and compared with other block and stream ciphers. Camellia-2a is the optimum implementation achieving a throughput of 18.41Mbps. Even though, our design was not optimized for Xilinx Spartan 2 FPGA family, we implemented it in order to compare it with

Table 3. Results for Compact camellia compared to other implementations of camellia, smallest implementation of other block ciphers and the eSTREAM Portfolio ciphers on FPGA

Design	Maximum Delay (ns)	Clock Cycles	Block Size (bits)	Key Size (bits)	Area (slices)	Block RAMs	Throughput (Mbps)	Throughput/ Area (Mbps/slice)	Device
Camellia -2a	7.95	875	128	128	318	0	18.41	0.06	xc3s50-5
Camellia -2b	9.37	875	128	128	214	2	15.61	0.07	xc3s50-5
Camellia -2a	11.01	875	128	128	399	0	13.28	0.03	xc2s30-6
Camellia -2b	18.72	875	128	128	282	3	7.82	0.03	xc2s30-6
Camellia [1]	22.62	44	128	128	908	0	128.58	0.14	Xilinx VirtexE
Camellia [9]	18.28	18	128	128	1023	8	388.9	0.25	xc3s1000
Camellia [9]	17.34	18	128	128	1031	8	410.5	0.27	xc3s1000
AES 8-bit[6]	14.93	3900	128	128	124	2	2.2	0.01	xc2s15-6
AES [7]	16.67	46	128	128	222	3	166	0.32	xc2s30-6
TinyXTEA-1 [8]	13.87	240	64	128	266	0	19.22	0.07	xc3s50-5
TinyXTEA-3 [8]	15.97	112	64	128	254	0	35.78	0.14	xc3s50-5
Grain v1 [10]	5.10	1	1	80	44	0	196	4.45	xc3s50-5
Grain 128 [10]	5.10	1	1	128	50	0	196	3.92	xc3s50-5
MICKEY v2 [10]	4.29	1	1	80	115	0	233	2.03	xc3s50-5
MICKEY 128 [10]	4.48	1	1	128	176	0	223	1.27	xc3s50-5
Trivium [10]	4.17	1	1	80	50	0	240	4.80	xc3s50-5
Trivium (x64) [10]	4.74	1	64	80	344	0	13,504	39.26	xc3s400-5

the smallest AES implementations. Our implementation has a higher throughput and efficiency then AES [6] but it is outperformed by AES [7] which is designed for higher throughput. Camellia has a comparable throughput to TinyXTEA-1 [8] and it is only marginally larger.

5. CONCLUSION

Our Camellia implementation is mainly based on usage of shift registers and DRAMs for efficient memory implementation. Furthermore, using shift registers also removes the need of addressing. This type of architecture is applicable for ciphers which have key scheduling based on shifting. In low power applications, where higher throughput with minimum area is required, its a good alternative for AES.

6. REFERENCES

[1] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, "Camellia: A 128-bit block cipher suitable for multiple platforms – design and analysis," in *SAC 2000*, ser. LNCS, vol. 2012. Springer, 2001, pp. 39–56.

[2] *Final report of NESSIE*, NESSIE, April 2004, https://www.cosic.esat.kuleuven.be/nessie/Bookv015.pdf.

[3] J. Lu, J. Kim, N. Keller, and O. Dunkelman, "Improving the efficiency of impossible differential cryptanalysis of reduced Camellia and MISTY1," in *CT-RSA 2008*, ser. LNCS, T. Malkin, Ed., vol. 4964. Berlin: Springer-Verlag, April 2008, pp. 370–386.

[4] D. Lei, L. Chao, and K. Feng, "New observation on Camellia," in *ACM Symposium on Applied Computing 2006*, ser. LNCS, B.Preneel and S. Tavares, Eds., vol. 3897. Berlin: Springer-Verlag, February 2006, pp. 51–64.

[5] G. Jie and Z. Zhongya, "Improved collision attack on reduced round Camellia," in *CANS 2006*, ser. LNCS, D.Pointcheval, Y. Mu, and K.Chen, Eds., vol. 4301. Berlin: Springer-Verlag, November 2006, pp. 189–190.

[6] T. Good and M. Benaissa, "AES on FPGA from the fastest to the smallest." in *CHES 2005*, ser. LNCS, J. R. Rao and B. Sunar, Eds., vol. 3659. Springer, 2005, pp. 427–440.

[7] P. Chodowiec and K. Gaj, "Very compact FPGA implementation of the AES algorithm," in *CHES 2003*, ser. LNCS, vol. 2779. Springer, Sep. 2003, pp. 319–333.

[8] J.-P. Kaps, "Chai-tea, cryptographic hardware implementations of xTEA," in *INDOCRYPT 2008*, ser. LNCS, D. Chowdhury, V. Rijmen, and A. Das, Eds., vol. 5365. Heidelberg: Springer, Dec 2008, pp. 363–375.

[9] D. Denning, I. James, and D. Malachy, "Compact iterative FPGA camellia algorithm implementation," in *FPT 2004*, December 2004, pp. 311–314.

[10] D. Hwang, M. Chaney, S. Karanam, N. Ton, and K. Gaj, "Comparison of FPGA-targeted hardware implementations of eSTREAM stream cipher candidates," in *SASC 2008*, Feb 2008, pp. 151–162.