Available online at www.scholarsresearchlibrary.com

Scholars Research Library

Archives of Applied Science Research, 2013, 5 (3):270-277
(http://scholarsresearchlibrary.com/archive.html)

ISSN 0975-508X
CODEN (USA) AASRC9

FPGA implementations of the hummingbird cryptographic algorithm

Ashwani Sengar, Prince Nagar and Mayank Sharma

E & CE Department, Sharda University, Greater Noida, India

ABSTRACT

Hummingbird is a new ultra-lightweight crypto-graphic algorithm targeted for resource-constrained devices like RFID tags, smart cards, and wireless sensor nodes. In this paper, we design the algorithim using Dsp module, we describe efficient hardware implementations of a Hummingbird component in field-programmable gate array (FPGA) devices. We implement an encryption/decryption core on the low-cost Xilinx FPGA series Vertex-5 and compare our results with other reported lightweight block cipher implementations. Our experimental results highlight that in the context of low-cost FPGA implementation Hummingbird has favorable efficiency and low area requirements.

Key words*:* Lightweight cryptographic primitive, resource-constrained devices, FPGA implementations.

INTRODUCTION

Hummingbird is a recently proposed ultra-lightweight cryp-tographic algorithm targeted for low-cost smart devices like RFID tags, smart cards, and wireless sensor nodes [3]. It has a hybrid structure of block cipher and stream cipher and was developed with both lightweight software and lightweight hardware implementations for constrained devices in mind. Moreover, Hummingbird has been shown to be resistant to the most common attacks to block ciphers and stream ciphers including birthday attack, differential and linear cryptanalysis, structure attacks, algebraic attacks, cube attacks, etc. [3].

In practice, Hummingbird has been implemented across a wide range of different target platforms [3], [5]. Those imple-mentations demonstrate that Hummingbird provides efficient and flexible software solutions for various embedded applica-tions. However, the hardware performance of Hummingbird has not yet been investigated in detail. As a result, our main contribution in this paper is to close this gap and provide the first efficient hardware implementations of Humming-bird encryption/decryption cores on low-cost FPGAs. Our implementation results show that on the Vertex-5 XC5VLX30 FPGA device the speed optimized Hummingbird encryption core can achieve a throughput of 160:4 Mbps at the cost of 273 slices, whereas the encryption/decryption core can be implemented in 558 slices and operate at 128:8 Mbps.

II. THE HUMMINGBIRD CRYPTOGRAPHIC ALGORITHM

Hummingbird is neither a block cipher nor a stream cipher, but a *rotor machine* equipped with novel rotor-stepping rules. The design of Hummingbird is based on an elegant combi-nation of a block cipher and stream cipher with 16-bit block size, 256-bit key size, and 80-bit internal state. Figure 1(a) and Figure 1(b) illustrate the initialization and encryption processes of the Hummingbird cryptographic algorithm, re-spectively. Both initialization and encryption consist of four 16-bit block ciphers E_{ki} (i = 1; 2; 3; 4), four 16-bit internal state registers RSi (i = 1; 2; 3; 4), and a 16-stage Linear Shift Feedback Register (LFSR). Moreover, the 256-bit secret key K is divided into four 64-bit subkeys k_1; k_2; k_3 and k_4 which are used in the four block ciphers, respectively.

After a system initialization process as shown in Figure 1(a), a 16-bit plaintext block PT_i is encrypted by passing

four identical block ciphers E_{ki} (·) (i = 1; 2; 3; 4) in a consecutive manner, each of which is a typical substitution-permutation (SP) network with 16-bit block size and 64-bit key as shown in Figure 1(c). The block cipher consists of four regular rounds and a final round. The substitution layer is composed of four S-boxes with 4-bit inputs and 4-bit outputs as shown in Table I.

TABLE I FOUR S-BOXES IN HEXADECIMAL NOTATION

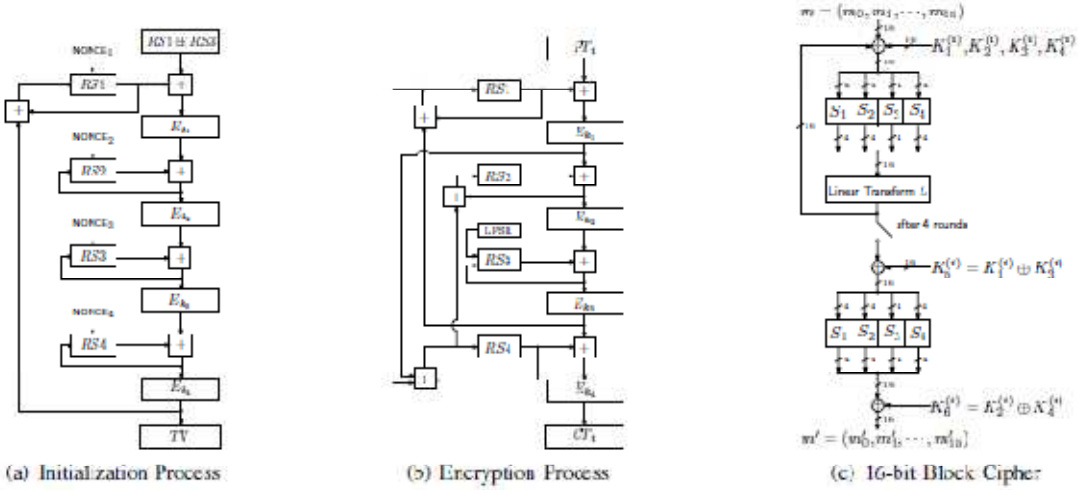
x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_1(x)$	8	6	5	F	1	C	A	9	E	B	2	4	7	0	D	3
$S_2(x)$	0	7	E	1	5	B	8	2	3	A	D	6	F	C	4	9
$S_3(x)$	2	E	F	5	C	1	9	A	B	4	6	8	0	7	3	D
$S_4(x)$	0	7	3	4	C	1	A	F	D	E	6	B	2	8	9	5

The permutation layer in the 16-bit block cipher is given by the linear transform $L : \{0; 1\}^{16} \rightarrow \{0; 1\}^{16}$ defined as follows:

$L(m) = m \oplus (m \lll 6) \oplus (m \lll 10)$;

where $m = (m_0; m_1; \cdots ; m_{15})$ is a 16-bit data block.

To further reduce the consumption of the area and power of Hummingbird in hardware implementations, four S-boxes used in Hummingbird can be replaced by a single S-box, which is repeated four times in the 16-bit block cipher. The compact version of Hummingbird can achieve the same security level as the original Hummingbird and will be implemented on FPGAs in this paper. For more details about Hummingbird, the interested reader is referred to [3].



(a) Initialization Process (b) Encryption Process (c) 16-bit Block Cipher

Fig. 1. The Hummingbird Cryptographic Algorithm and Its Internal Structure

TABLE II AREA REQUIREMENT COMPARISON FOR THE LOOP-UNROLLED ARCHITECTURE OF 16-BIT BLOCK CIPHER ON THE SPARTAN-3 XC3S200 FPGA

S-box		Implementation Strategy	# LUTs	# FFs	Total Occupied Slices	
S_1	(x)	LUT	186	16	107	
		BFR	186	16	109	
S_2	(x)	LUT	193	16	112	
		BFR	186	16	107	
S_3	(x)	LUT	186	16	101	
		BFR	186	16	106	
S_4	(x)	LUT	190	16	104	
		BFR	187	16	109	

When comparing different S-boxes and implementation strategies, Table II shows that the loop-unrolled architecture occupies the minimal number of slices provided that the S-box $S_3(x)$ is employed and implemented by a LUT. Therefore, the S-box $S_3(x)$ is chosen for efficient implementation of speed optimized Hummingbird

encryption/decryption cores that are described in detail in the following subsections.

Table III Notation

PT_i	the i-th 16-bit plaintext block, $i = 1, 2, \ldots, n$
CT_i	the i-th 16-bit ciphertext block, $i = 1, 2, \ldots, n$
K	the 256-bit secret key
$\mathbf{E}_K(\cdot)$	the encryption function of Hummingbird with 256-bit secret key K
$\mathbf{D}_K(\cdot)$	the decryption function of Hummingbird with 256-bit secret key K
k_i	the 64-bit subkey used in the i-th block cipher, $i = 1, 2, 3, 4$, such that $K = k_1 \| k_2 \| k_3 \| k_4$
$E_{k_i}(\cdot)$	a block cipher encryption algorithm with 16-bit input, 64-bit key k_i, and 16-bit output, i.e., $E_{k_i} : \{0,1\}^{16} \times \{0,1\}^{64} \rightarrow \{0,1\}^{16}, i = 1, 2, 3, 4$
$D_{k_i}(\cdot)$	a block cipher decryption algorithm with 16-bit input, 64-bit key k_i, and 16-bit output, i.e., $D_{k_i} : \{0,1\}^{16} \times \{0,1\}^{64} \rightarrow \{0,1\}^{16}, i = 1, 2, 3, 4$
RSi	the i-th 16-bit internal state register, $i = 1, 2, 3, 4$
LFSR	a 16-stage Linear Feedback Shift Register with the characteristic polynomial $f(x) = x^{16} + x^{15} + x^{12} + x^{10} + x^7 + x^3 + 1$
$\boxplus$	modulo 2^{16} addition operator
$\boxminus$	modulo 2^{16} subtraction operator
$\oplus$	exclusive-OR (XOR) operator
$m \lll l$	left circular shift operator, which rotates all bits of m to the left by l bits, as if the left and the right ends of m were joined.
$K_j^{(i)}$	the j-th 16-bit key used in the i-th block cipher, $j = 1, 2, 3, 4$, such that $k_i = K_1^{(i)} \| K_2^{(i)} \| K_3^{(i)} \| K_4^{(i)}$
$S_i(x)$	the i-th 4-bit to 4-bit S-box used in the block cipher, $S_i(x) : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4, i = 1, 2, 3, 4$
NONCE_i	the i-th nonce which is a 16-bit random number, $i = 1, 2, 3, 4$
IV	the 64-bit initial vector, such that $IV = \text{NONCE}_1 \| \text{NONCE}_2 \| \text{NONCE}_3 \| \text{NONCE}_4$

A . Initialization Process

The overall structure of the Hummingbird initialization algorithm is shown in Figure 1(a). When using Hummingbird in practice, four 16-bit random nonces NONCE_i are first chosen to initialize the four internal state registers *RSi* (*i* = 1*;* 2*;* 3*;* 4), respectively, followed by four consecutive encryptions on the message *RS*1 *RS*3 by Hummingbird running in initialization mode (see Figure 1(a)). The final 16-bit ciphertext *TV* is used to initialize the LFSR. Moreover, the 13th bit of the LFSR is always set to prevent a zero register. The LFSR is also stepped once before it is used to update the internal state register *RS*3. We summarize the Hummingbird initialization process in the following Algorithm 1.

Algorithm 1 Hummingbird Initialization
Input: Four 16-bit random nonce NONCE_i (*i* = 1*;* 2*;* 3*;* 4)
Output: Initialized four rotors RSi_4 (*i* = 1*;* 2*;* 3*;* 4) and LFSR

1: $RS1_0$ = NONCE_1 [Nonce Initialization]
2: $RS2_0$ = NONCE_2
3: $RS3_0$ = NONCE_3
4: $RS4_0$ = NONCE_4
5: **for** *t* = 0 to 3 **do**
6: $V12_t = E_{k1}((RS1_t \quad RS3_t) \quad RS1_t)$
7: $V23_t = E_{k2}(V12_t \quad RS2_t)$
8: $V34_t = E_{k3}(V23_t \quad RS3_t)$
9: $TV_t = E_{k4}(V34_t \quad RS4_t)$
10: $RS1_{t+1} = RS1_t \quad TV_t$
11: $RS2_{t+1} = RS2_t \quad V12_t$
12: $RS3_{t+1} = RS3_t \quad V23_t$
13: $RS4_{t+1} = RS4_t \quad V34_t$
14: **end for**
15: LFSR = TV_3 | 0x1000 [LFSR Initialization]
16: **return** RSi_4 (*i* = 1*;* 2*;* 3*;* 4) and LFSR

B . Encryption Process

The overall structure of the Hummingbird encryption algorithm is depicted in Figure 1(b). After a system initialization process, a 16-bit plaintext block PT_i is encrypted by first exe-cuting a modulo 2^{16} addition of PT_i and the content of the first internal state register *RS*1. The result of the addition is then encrypted by the first block cipher E_{k1} . This procedure is repeated in a similar manner for another three times and the output of E_{k4} is the correspond-ing ciphertext CT_i. Furthermore, the states of the four internal state registers will also be updated in an

unpredictable way based on their current states, the outputs of the first three block ciphers, and the state of the LFSR. Algorithm 2 describes the detailed procedure of Hummingbird encryption.

Algorithm 2 Hummingbird Encryption
Input: A 16-bit plaintext PT_i and four rotors RSi_t ($i = 1; 2; 3; 4$)
Output: A 16-bit ciphertext CT_i

1: $V12_t = E_{k1}(PT_i \quad RS1_t)$ [Block Encryption]
2: $V23_t = E_{k2}(V12_t \quad RS2_t)$
3: $V34_t = E_{k3}(V23_t \quad RS3_t)$
4: $CT_i = E_{k4}(V34_t \quad RS4_t)$
5: $\text{LFSR}_{t+1} \leftarrow \text{LFSR}_t$ [Internal State Updating]
6: $RS1_{t+1} = RS1_t \quad V34_t$
7: $RS3_{t+1} = RS3_t \quad V23_t \quad \text{LFSR}_{t+1}$
8: $RS4_{t+1} = RS4_t \quad V12_t \quad RS1_{t+1}$
9: $RS2_{t+1} = RS2_t \quad V12_t \quad RS4_{t+1}$
10: **return** CT_i

C . Decryption Process
The overall structure of the Hummingbird decryption algorithm is illustrated in Figure 1(c). The decryption process follows the similar pattern as the encryption and a detailed descrip-tion is shown in the following Algorithm 3.

Algorithm 3 Hummingbird Decryption
Input: A 16-bit ciphertext CT_i and four rotors RSi_t ($i = 1; 2; 3; 4$)
Output: A 16-bit plaintext PT_i

1: $V34_t = D_{k4}(CT_i) \quad RS4_t$ [Block Decryption]
2: $V23_t = D_{k3}(V34_t) \quad RS3_t$
3: $V12_t = D_{k2}(V23_t) \quad RS2_t$
4: $PT_i = D_{k1}(V12_t) \quad RS1_t$
5: $\text{LFSR}_{t+1} \leftarrow \text{LFSR}_t$ [Internal State Updating]
6: $RS1_{t+1} = RS1_t \quad V34_t$
7: $RS3_{t+1} = RS3_t \quad V23_t \quad \text{LFSR}_{t+1}$
8: $RS4_{t+1} = RS4_t \quad V12_t \quad RS1_{t+1}$
9: $RS2_{t+1} = RS2_t \quad V12_t \quad RS4_{t+1}$
10: **return** PT_i

D. 16-Bit Block Cipher
Hummingbird employs four identical block ciphers $E_{ki}(\cdot)$ ($i = 1; 2; 3; 4$) in a consecutive manner, each of which is a typical substitution-permutation (SP) network with 16-bit block size and 64-bit key as shown in the following Figure 2.

The block cipher consists of four regular rounds and a final round. The 64-bit subkey k_i is split into four 16-bit round keys $K_1^{(i)}; K_2^{(i)}; K_3^{(i)}$ and $K_4^{(i)}$ that are used in the four regular rounds, respectively. Moreover, the final round utilizes two keys $K_5^{(i)}$ and $K_6^{(i)}$ directly derived from the four round keys (see Fig. 2). While each regular round comprises of a key mixing step, a substitution layer, and a permutation layer, the final round only includes the key mixing and the S-box substitution steps. The key mixing step is implemented using a simple exclusive-OR operation, whereas the substitution layer is composed of four S-boxes with 4-bit inputs and 4-bit outputs as shown in Table IV.

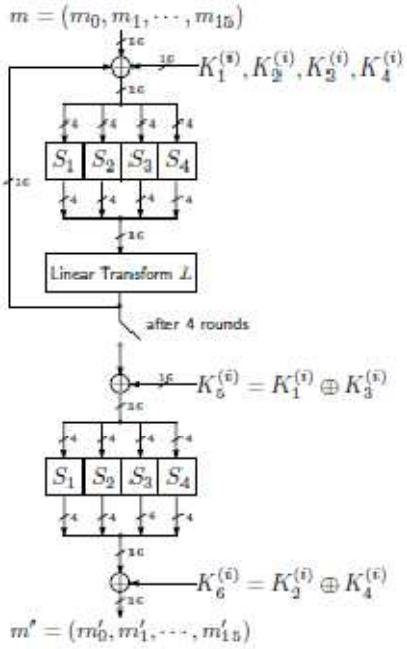


Fig. 2The Structure of Block Cipher in the Hummingbird Cryptographic Algorithm

Table IV Four S-Boxes in Hexadecimal Notation

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_1(x)$	8	6	5	F	1	C	A	9	E	B	2	4	7	0	D	3
$S_2(x)$	0	7	E	1	5	B	8	2	3	A	D	6	F	C	4	9
$S_3(x)$	2	E	F	5	C	1	9	A	B	4	6	8	0	7	3	D
$S_4(x)$	0	7	3	4	C	1	A	F	D	E	6	B	2	8	9	5

The selected four S-boxes, denoted by $S_i(x) : F^4_2 \rightarrow F^4_2$; $i = 1; 2; 3; 4$, are Serpent-type S-boxes [1] with additional properties (see [9] for more details) which can ensure that the 16-bit block cipher is resistant to linear and differential attacks as well as interpolation attack. The permutation layer in the 16-bit block cipher is given by the linear transform
$L : \{0; 1\}^{16} \rightarrow \{0; 1\}^{16}$ defined as follows:

$L(m) = m \square (m \lll 6) \oplus (m \lll 10);$

where $m = (m_0; m_1; \cdots ; m_{15})$ is a 16-bit data block. We give a detailed description for the encryption process of the 16-bit block cipher in the following Algorithm 4. The decryption process can be easily derived from the encryption and therefore is omitted here.

III. FPGA IMPLEMENTATIONS OF HUMMINGBIRD

In this section efficient FPGA implementations of a stand-alone Hummingbird component are described. Note that the choice of different kinds of I/O interfaces has a significant influence on the performance of hardware implementation and is highly application specific. Therefore, we do not implement any specific I/O logic in order to obtain the accurate perfor-mance profile of a plain Hummingbird encryption/decryption core and to provide enough flexibility for various applications.

A. Selection of a "Hardware-Friendly" S-Box

A "hardware-friendly" S-box is the S-box that can be efficiently implemented in the target hardware platform with a small area requirement. Four 4 × 4 S-boxes $S_i(x) : F^4_2 \rightarrow F^4_2$ ($i = 1; 2; 3; 4$) have been carefully selected in Hummingbird according to certain security criteria (see Section II). To implement the compact version of Hummingbird, we need to choose a "hardware-friendly" S-box from four S-boxes listed in Table I. By using the Boolean minimization tool *Espresso* [4] we can obtain the minimal Boolean function representa-tions (BFR) for the four S-boxes in Hummingbird. Note that each S-box can be implemented in hardware by using either a look-up table (LUT) or the Boolean function representations (i.e., combinatorial logic). The exact efficiency of the above two approaches significantly depends on specific hardware platforms and synthesis tools. Therefore, for the proposed architecture of the 16-bit block cipher in Section III-B we investigate two implementation strategies (i.e., LUT and BFR) for the four S-boxes and select one that results in the most area-efficient implementation of the 16-bit block

cipher.

B. Loop-Unrolled Architecture of 16-bit Block Cipher

The loop-unrolled architecture for the 16-bit block cipher is illustrated in Figure 3. In this architecture, only one 16-bit block of data is processed at a time. However, five rounds are cascaded and the whole encryption can be performed in a single clock cycle. The loop-unrolled architecture consists of 8 XORs, 20 S-boxes, and 4 permutation layers for the datapath. To select a "hardware-friendly" S-box for the compact version of Hummingbird, we implement the loop-unrolled architec-ture of the 16-bit block cipher on the target FPGA platform and test one S-box candidate from Table I each time. Table II summarizes the area requirement when using different S-boxes and implementation strategies. All experimental results are from post-place and route analysis.

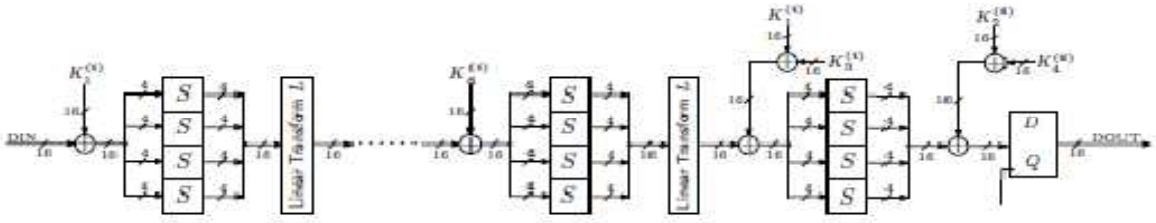


Fig. 3. Loop-Unrolled Architecture of 16-bit Block Cipher

TABLE V AREA REQUIREMENT COMPARISON FOR THE LOOP-UNROLLED ARCHITECTURE OF 16-BIT BLOCK CIPHER ON THE SPARTAN-3 XC3S200 FPGA

S-box		Implementation Strategy	# LUTs	# FFs	Total Occupied Slices	
S_1	(x)	LUT	186	16	107	
		BFR	186	16	109	
S_2	(x)	LUT	193	16	112	
		BFR	186	16	107	
S_3	(x)	LUT	186	16	101	
		BFR	186	16	106	
S_4	(x)	LUT	190	16	104	
		BFR	187	16	109	

When comparing different S-boxes and implementation strategies, Table V shows that the loop-unrolled architecture occupies the minimal number of slices provided that the S-box $S_3(x)$ is employed and implemented by a LUT. Therefore, the S-box $S_3(x)$ is chosen for efficient implementation of speed optimized Hummingbird encryption/decryption cores that are described in detail in the following subsections.

C. Speed Optimized Hummingbird Encryption Core

The top-level description of a speed optimized Humming-bird encryption core is illustrated in Figure 3. After the chip enable signal changes from '0' to '1', the initialization process (see Figure 1(a)) begins and four rotors *RSi* (i = 1*;* 2*;* 3*;* 4) are first initialized by four 16-bit random nonce through the interface *RSi* within four clock cycles. From the fifth clock cycle, the core starts encrypting *RS*1 *RS*3 for four times and each iteration requires four clock cycles to finish encryptions by four 16-bit block ciphers as well as the internal state updating. During the above procedure, the 64-bit subkeys k_i (i = 1*;* 2*;* 3*;* 4) are read from an external register under the control of a key selection signal. Moreover, depending on the value of a round counter, the multiplexer M_5 chooses the correct computation results to update four rotors and other multiplexers select appropriate inputs to feed the 16-bit block cipher. Once the initialization process is done after 20 clock cycles, the first 16-bit plaintext block is read from an external register for encryption. With another four clock cycles, the corresponding ciphertext is output from the encryption core. Therefore, the proposed speed optimized Hummingbird en-cryption core can encrypt one 16-bit plaintext block per 4 clock cycles, after an initialization process of 20 clock cycles.

D. Implementation Results and Comparisons

A summary of our implementation results is presented in Table VI, where the area requirements (in slices), the maximum work frequency, and the throughput are provided. All experimental results were extracted after place and route with the ISE Design Suite 9.2i from Xilinx on a XC5VLX30 Vertex-5 platform with speed grade −5. From Table III, we note that the speed optimized Hummingbird encryption core can achieve a throughput of 160*:*4 Mbps at the cost of 273 slices, whereas the Hummingbird encryption/decryption core occupies 558 slices and operates at 128*:*8 Mbps on the target FPGA platform.

TABLE VI IMPLEMENTATION RESULTS FOR COMPACT VERSION OF HUMMINGBIRD ON

THE XC5VLX30 Vertex-5 FPGA

Mode	# LUTs	# FFs	Total Occupied	Max. Freq.	# CLK Cycles		Throughput	Efficiency
(Enc/Dec)			Slices	(MHz)	Init.	Enc/Dec	(Mbps)	(Mbps/# Slices)
Enc	473	120	273	40:1	20	4	160:4	0:59
Enc/Dec	1; 024	145	558	32:2			128:8	0:23

TABLE VII PERFORMANCE COMPARISON OF FPGA IMPLEMENTATIONS OF CRYPTOGRAPHIC ALGORITHMS

Cipher	Key	Block	FPGA	Total Occupied	Max. Freq.	Throughput	Efficiency
	Size	Size	Device	Slices	(MHz)	(Mbps)	(Mbps/# Slices)
Hummingbird	256	16	Vertex-5 XC5VLX30	273	40:1	160:4	0:59
PRESENT [10]	80	64	Spartan-3 XC3S400-5	176	258	516	2:93
	128	64		202	254	508	2:51
PRESENT [7]	80	64	Spartan-3E XC3S500	271	–	–	–
XTEA [8]	128	64	Spartan-3 XC3S50-5	254	62:6	36	0:14
			Virtex-5 XC5VLX85-3	9; 647	332:2	20; 645	2:14
ICEBERG [12]	128	64	Virtex-2	631	–	1; 016	1:61
SEA [9]	126	126	Virtex-2 XC2V4000	424	145	156	0:368
AES [2]			Spartan-2 XC2S30-6	522	60	166	0:32
AES [6]			Spartan-3 XC3S2000-5	17; 425	196:1	25; 107	1:44
	128	128	Spartan-2 XC2S15-6	264	67	2:2	0:01
AES [11]			Spartan-2 XC2V40-6	1; 214	123	358	0:29
AES [1]			Spartan-3	1; 800	150	1700	0:9

Table VII describes the performance comparison of our Hummingbird implementation with existing FPGA implementations of block ciphers PRESENT [7], [10], XTEA [8], ICEBERG [12], SEA [9] as well as AES [1], [2], [6], [11]. Note that numerous AES hardware architectures have been proposed in literature and we only focus on those implementations using low-cost Vertex series FPGA devices with speed grade -5 and above for the purpose of comparison. Moreover, the implementation figures of ICEBERG and SEA are only available on Virtex-2 series FPGAs. We also would like to point out that it is quite difficult to provide a fair com-parison among different implementations on FPGAs, taking into account the diversity of FPGA devices and packages, speed grade level, and synthesis and implementation tools. Therefore, we also include additional information such as implementation platform and speed grade level in Table IV.

Our experimental results show that in the context of low-cost FPGA implementation Hummingbird can achieve larger throughput with smaller area requirement, when compared to block ciphers XTEA, ICEBERG, SEA and AES. How-ever, the implementation of the ultra-lightweight block cipher PRESENT is more efficient than that of Hummingbird, although a slightly larger (and hence more expensive) FPGA device Vertex-5 XC5VLX30 is required. The main reason is due to the complex internal state updating procedure in Hummingbird cipher (see Figure 1(a) and Figure 1(b)). As a result, the control unit is more complicated and the delay of the critical path is much longer in the Hummingbird hardware architecture than those in the PRESENT core.

CONCLUSION

This paper presented the first efficient FPGA implementa-tions of the ultra-lightweight cryptographic algorithm Hum-mingbird. The proposed speed optimized Hummingbird encryption/decryption cores can encrypt or decrypt a 16-bit message block with 4 clock cycles, after an initialization process of 20 clock cycles. Compared to other lightweight FPGA implementations of block ciphers XTEA, ICEBERG, SEA and AES, Hummingbird can achieve larger throughput with smaller area requirement. Consequently, Hummingbird can be considered as an ideal cryptographic primitive for resource-constrained environments.

REFERENCES

[1] P. Bulens, F.-X. Standaert, J.-J. Quisquater, and P. Pellegrin, *Progress in Cryptology - AFRICACRYPT* **2008**, LNCS 5023, pp. 16-26, 2008.
[2] P. Chodowiec and K. Gaj, “Very Compact FPGA Implementation of the AES Algorithm”, *The 5th International Workshop on Cryptographic Hardware and Embedded Systems - CHES* **2003**, LNCS 2779, pp. 319-333, 2003.
[3] D. Engels, X. Fan, G. Gong, H. Hu, and E. M. Smith, “Hummingbird: Ultra-Lightweight Cryptography for Resource- Constrained Devices”, to appear in the proceedings of *The 14th International Conference on Financial Cryptography and Data Security - FC 2010*, **2010**.
[4] N. N. Espresso. Available at http://embedded.eecs.berkeley.edu/pubs/ downloads/espresso/index.htm, November

1994.
[5] X. Fan, H. Hu, G. Gong, E. M. Smith and D. Engels, "Lightweight Implementation of Hummingbird Cryptographic Algorithm on 4-Bit Microcontrollers", *The 1st International Workshop on RFID Security and Cryptography 2009 (RISC'09)*, pp. 838-844, **2009**.
[6] T. Good and M. Benaissa, "AES on FPGA from the Fastest to the Smallest", *The 7th International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2005*, LNCS 3659, pp. 427-440, **2005**.
[7] X. Guo, Z. Chen, and P. Schaumont, "Energy and Performance Evaluation of an FPGA-Based SoC Platform with AES and PRESENT Copro-cessors", *Embedded Computer Systems: Architectures, Modeling, and Simulation - SAMOS'2008*, LNCS 5114, pp. 106-115, **2008**.
[8] J.-P. Kaps, "Chai-Tea, Cryptographic Hardware Implemenations of xTEA", *The 9th International Conference on Cryptology in India - INDOCRYPT 2008*, LNCS 5356, pp. 363-375, **2008**.
[9] F. Mace, F.-X. Standaert, and J.-J. Quisquater, "FPGA Implemenation(s) of a Scalable Encryption Algorithm", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 212-216, **2008**.
[10] A. Poschmann, "Lightweight Cryptography - Cryptographic Engineering for a Pervasive World", Ph.D. Thesis, Department of Electrical Engi-neering and Information Sciences, Ruhr-Universit*a*•et Bochum, Bochum, Germany, **2009**.
[11] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat, "Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Applica-tions", *International Conference on Information Technology: Coding and Computing - ITCC 2004*, pp. 583-587, **2004**.
[12] F.-X. Standaert, G. Piret, G. Rouvroy, and J.-J. Quisquater, "FPGA Implementations of the ICEBERG Block Cipher", *Integration, the VLSI Journal*, vol. 40, iss. 1, pp. 20-27, **2007**.
[13] Xilinx Inc., "Spartan-3 FPGA Family Data Sheet", DS099, December 4, **2009**, available at http://www.xilinx.com/support/documentation/data sheets/ds099.pdf.