Design Space Exploration of Hummingbird Implementations on FPGAs

Xinxin Fan and Guang Gong
E&CE Department
University of Waterloo
Waterloo, Ontario, N2L 3G1, CANADA
Email: {x5fan, ggong}@uwaterloo.ca

Ken Lauffenburger
Aava Technology LLC
1206 Donegal Ln
Garland, TX 75044, USA
Email: kenl@aavatech.com

Troy Hicks
Revere Security Corporation
4500 Westgrove Drive, Suite 335
Addison, TX 75001, USA
Email: troy.hicks@reveresecurity.com

***Abstract*—Hummingbird is a recently proposed ultra-lightweight cryptographic algorithm targeted for resource-constrained devices like RFID tags, smart cards, and wireless sensor nodes. In this paper, we describe efficient hardware implementations of a stand-alone Hummingbird component in field-programmable gate array (FPGA) devices. We implement an encryption only core and an encryption/decryption core on the low-cost Xilinx FPGA series Spartan-3 and compare our results with other reported lightweight block cipher implementations on the same series. Moreover, a speed-optimized and an area-optimized hardware architectures are also proposed in this contribution. Our experimental results highlight that in the context of low-cost FPGA implementation Hummingbird has favorable efficiency and low area requirements.**

***Index Terms*—Lightweight cryptographic primitive, resource-constrained devices, FPGA implementations.**

I. Introduction

The widespread deployment of various wireless networks such as mobile ad-hoc networks, sensor networks, mesh networks, personal area networks and RFID systems is making possible a world of pervasive computing a reality. While the wireless communication technology and devices under development are enabling our march toward the era of pervasive computing, the security and privacy concerns in pervasive computing remains a serious impediment to widespread adoption of emerging technologies. Employing cryptographic primitives to perform strong authentication and encryption and provide other security functionalities is a promising solution to overcome those concerns.

For many years, the cryptographic engineering communities had worked on the problem of implementing various cryptographic primitives as fast as possible. Typical examples were high-speed RSA and Advanced Encryption Standard (AES) engines. However, the upcoming pervasive computing era that features myriads of small, inexpensive, robust networked processing devices has put forward the new challenge to the implementation of security mechanisms for embedded applications. Ultra low-cost smart devices such as RFID tags, smart cards, and wireless sensor nodes usually have extremely constrained resources in terms of computational capabilities, memory, and power supply. Consequently, classical cryptographic primitives designed for full-fledged computers might not be suited for resource-constrained pervasive devices and it is often desirable to have cryptographic primitives as small as possible. As a response to the aforementioned issue, *lightweight cryptography*, which focuses on designing new cryptographic primitives with small footprint in hardware and low average and peak power consumption, has received a lot of attention from both academia and industry in recent years.

The key issue of designing lightweight cryptographic algorithms is to deal with the trade-off among *security*, *cost*, and *performance* and find an optimal cost-performance ratio [24]. Quite a few lightweight symmetric ciphers that particularly target resource-constrained smart devices have been published in the past few years and those ciphers can be utilized as basic building blocks to design security mechanisms for embedded applications. All the previous proposals can be roughly divided into the following three categories. The first category consists of highly optimized and compact hardware implementations for standardized block ciphers such as AES [12], [13], [16], IDEA [22] and XTEA [19], whereas the proposals in the second category involve slight modifications of a classical block cipher like DES [20] for lightweight applications. Finally, the third category features new low-cost designs, including lightweight block ciphers HIGHT [18], mCrypton [21], SEA [26], PRESENT [3] and KATAN and KTANTAN [5], as well as lightweight stream ciphers Grain [17], Trivium [6] and MICKEY [2]. A good survey covering recently published lightweight cryptographic implementations can be found in [8].

Hummingbird is a recently proposed ultra-lightweight cryptographic algorithm targeted for low-cost smart devices [9]. 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. The hybrid model can provide the designed security with small block size and is therefore expected to meet the stringent response time and power consumption requirements for a large variety of embedded applications. 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. [9].

In practice, Hummingbird has been implemented across a

wide range of different software platforms, including the 4-bit microcontroller ATAM893-D [11] and the 8-bit microcontroller ATmega128L from Atmel as well as the 16-bit microcontroller MSP430 from Texas Instrument (TI) [9]. Those implementations demonstrate that Hummingbird provides efficient and flexible software solutions for various embedded applications. 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 small, power and energy efficient implementations of Hummingbird encryption/decryption cores on low-cost FPGAs. Our implementation results show that on the Spartan-3 XC3S200 FPGA device the speed-optimized Hummingbird encryption core can achieve a throughput of 160.4 Mbps at the cost of 273 slices, whereas the area-optimized encryption core can be implemented in 253 slices and operate at 66.1 Mbps. Furthermore, for applications requiring both encryption and decryption, our speed-optimized and area-optimized Hummingbird encryption/decryption cores can achieve throughput rates of 128.8 and 61.4 Mbps and occupy 558 and 363 slices on the target FPGA platform, respectively.

The remainder of this paper is organized as follows. Section II gives a brief description of the Hummingbird cryptographic algorithm. Subsequently, in Section III the hardware architectures of speed-optimized and area-optimized Hummingbird encryption/decryption cores are described and our implementation results are presented and compared with other lightweight block cipher implementations on the similar FPGA platforms. Finally, Section IV concludes this contribution.

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 combination of block cipher and stream cipher with 16-bit block size, 256-bit key size, and 80-bit internal state. The size of the key and the internal state of Hummingbird provides a security level which is adequate for many embedded applications. For clarity, we use the notation listed in Table I in the algorithm description. A top-level structure of the Hummingbird cryptographic algorithm is shown in Figure 1, which consists of four 16-bit block ciphers E_{k_i} or D_{k_i} $(i = 1, 2, 3, 4)$, 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.

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 $RS1 \boxplus RS3$ by Hummingbird running in initialization mode (see Figure 1(a)). The final 16-bit ciphertext TV is used to initialize the LFSR. Moreover, the 13^{th} 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 $RS3$. 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

$RS1_0 = \mathsf{NONCE}_1$ [Nonce Initialization]
$RS2_0 = \mathsf{NONCE}_2$
$RS3_0 = \mathsf{NONCE}_3$
$RS4_0 = \mathsf{NONCE}_4$
for $t = 0$ to 3 **do**
  $V12_t = E_{k_1}((RS1_t \boxplus RS3_t) \boxplus RS1_t)$
  $V23_t = E_{k_2}(V12_t \boxplus RS2_t)$
  $V34_t = E_{k_3}(V23_t \boxplus RS3_t)$
  $TV_t = E_{k_4}(V34_t \boxplus RS4_t)$
  $RS1_{t+1} = RS1_t \boxplus TV_t$
  $RS2_{t+1} = RS2_t \boxplus V12_t$
  $RS3_{t+1} = RS3_t \boxplus V23_t$
  $RS4_{t+1} = RS4_t \boxplus V34_t$
end for
LFSR = $TV_3 \mid$ 0x1000 [LFSR Initialization]
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 executing a modulo 2^{16} addition of PT_i and the content of the first internal state register $RS1$. The result of the addition is then encrypted by the first block cipher E_{k_1}. This procedure is repeated in a similar manner for another three times and the output of E_{k_4} is the corresponding 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.

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 description is shown in the following Algorithm 3.

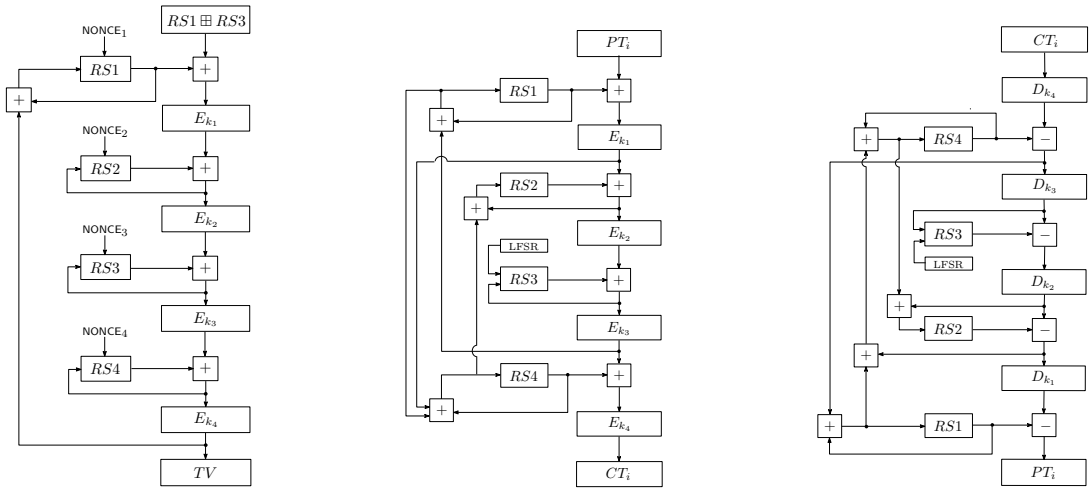
D. 16-Bit Block Cipher

Hummingbird employs four identical block ciphers $E_{k_i}(\cdot)$ $(i = 1, 2, 3, 4)$ 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

TABLE I
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., $E_{k_i} : \{0,1\}^{16} \times \{0,1\}^{64} \to \{0,1\}^{16}, i = 1, 2, 3, 4$
$D_{k_i}(\cdot)$	a block cipher decryption algorithm with 16-bit input, 64-bit key k_i, and 16-bit output, i.e., $D_{k_i} : \{0,1\}^{16} \times \{0,1\}^{64} \to \{0,1\}^{16}, i = 1, 2, 3, 4$
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, $S_i(x) : \mathbb{F}_2^4 \to \mathbb{F}_2^4, i = 1, 2, 3, 4$
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 $IV = \mathsf{NONCE}_1 \| \mathsf{NONCE}_2 \| \mathsf{NONCE}_3 \| \mathsf{NONCE}_4$



(a) Initialization Process (b) Encryption Process (c) Decryption Process

Fig. 1. A Top-Level Description of the Hummingbird Cryptographic Algorithm

TABLE II
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

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

$V12_t = E_{k_1}(PT_i \boxplus RS1_t)$ [Block Encryption]
$V23_t = E_{k_2}(V12_t \boxplus RS2_t)$
$V34_t = E_{k_3}(V23_t \boxplus RS3_t)$
$CT_i = E_{k_4}(V34_t \boxplus RS4_t)$
$\text{LFSR}_{t+1} \leftarrow \text{LFSR}_t$ [Internal State Updating]
$RS1_{t+1} = RS1_t \boxplus V34_t$
$RS3_{t+1} = RS3_t \boxplus V23_t \boxplus \text{LFSR}_{t+1}$
$RS4_{t+1} = RS4_t \boxplus V12_t \boxplus RS1_{t+1}$
$RS2_{t+1} = RS2_t \boxplus V12_t \boxplus RS4_{t+1}$
return CT_i

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

$V34_t = D_{k_4}(CT_i) \boxminus RS4_t$ [Block Decryption]
$V23_t = D_{k_3}(V34_t) \boxminus RS3_t$
$V12_t = D_{k_2}(V23_t) \boxminus RS2_t$
$PT_i = D_{k_1}(V12_t) \boxminus RS1_t$
$\text{LFSR}_{t+1} \leftarrow \text{LFSR}_t$ [Internal State Updating]
$RS1_{t+1} = RS1_t \boxplus V34_t$
$RS3_{t+1} = RS3_t \boxplus V23_t \boxplus \text{LFSR}_{t+1}$
$RS4_{t+1} = RS4_t \boxplus V12_t \boxplus RS1_{t+1}$
$RS2_{t+1} = RS2_t \boxplus V12_t \boxplus RS4_{t+1}$
return PT_i

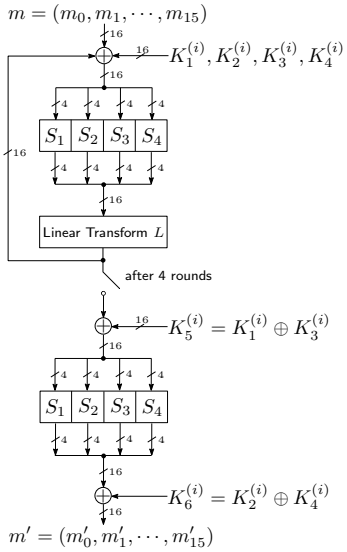


Fig. 2. The Structure of Block Cipher in the Hummingbird Cryptographic Algorithm

$K_1^{(i)}, K_2^{(i)}, K_3^{(i)}$ 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 II.

The selected four S-boxes, denoted by $S_i(x) : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4, i = 1, 2, 3, 4$, 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:

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

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.

Algorithm 4 16-bit Block Cipher Encryption $E_{k_i}(\cdot)$

Input: A 16-bit data block $m = (m_0, m_1, \cdots, m_{15})$ and a 64-bit subkey k_i such that
subkey $k_i = K_1^{(i)} \| K_2^{(i)} \| K_3^{(i)} \| K_4^{(i)}$

Output: A 16-bit date block $m' = (m'_0, m'_1, \cdots, m'_{15})$

for $j = 1$ to 4 **do**
  $m \leftarrow m \oplus K_j^{(i)}$ [key mixing step]
  $A = m_0 \| m_1 \| m_2 \| m_3, \quad B = m_4 \| m_5 \| m_6 \| m_7$
  $C = m_8 \| m_9 \| m_{10} \| m_{11}, D = m_{12} \| m_{13} \| m_{14} \| m_{15}$
  $m \leftarrow S_1(A) \| S_2(B) \| S_3(C) \| S_4(D)$ [substitution layer]
  $m \leftarrow m \oplus (m \lll 6) \oplus (m \lll 10)$ [permutation layer]
end for
$m \leftarrow m \oplus K_1^{(i)} \oplus K_3^{(i)}$
$A = m_0 \| m_1 \| m_2 \| m_3, \quad B = m_4 \| m_5 \| m_6 \| m_7$
$C = m_8 \| m_9 \| m_{10} \| m_{11}, D = m_{12} \| m_{13} \| m_{14} \| m_{15}$
$m \leftarrow S_1(A) \| S_2(B) \| S_3(C) \| S_4(D)$
$m' \leftarrow m \oplus K_2^{(i)} \oplus K_4^{(i)}$
return $m' = (m'_0, m'_1, \cdots, m'_{15})$

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.

III. FPGA Implementations of Hummingbird Cipher

In this section efficient FPGA implementations of a stand-alone Hummingbird component are described. We imple-

ment an encryption only core and an encryption/decryption core on the low-cost Xilinx FPGA series Spartan-3 and compare our results with other reported (ultra-)lightweight block cipher implementations on the same series. Moreover, a speed-optimized and an area-optimized hardware architectures are also described in this section. 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 performance profile of a plain Hummingbird encryption/decryption core as well as provide enough flexibility for various applications.

A. Target Platform and Design Tools

FPGAs are composed of configurable logic blocks (CLB) and a programmable interconnection network. We implement both encryption and decryption modules in VHDL for the low-cost Spartan-3 XC3S200 (Package FT256 with speed grade -5) FPGA device from Xilinx[1] [28]. We use the integrated FPGA development environment Aldec Active-HDL 8.2sp1 for writing, debugging and simulating VHDL codes. Furthermore, Synopsys Synplify Pro C-2009.06-SP1 and Xilinx ISE Design Suite v11.1 are employed for the design synthesis and implementation, respectively.

B. 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 $S_i(x) : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$ $(i = 1, 2, 3, 4)$ have been carefully selected in Hummingbird according to certain security criteria (see Section II-D). To implement the compact version of Hummingbird, we need to choose a "hardware-friendly" S-box from four S-boxes listed in Table II. Let $x = (x_3 \| x_2 \| x_1 \| x_0)$ be the 4-bit input to the S-box and let $S_i(x) = (S_i^{(3)}(x) \| S_i^{(2)}(x) \| S_i^{(1)}(x) \| S_i^{(0)}(x))$ denote the 4-bit output of the i-th S-box $(i = 1, 2, 3, 4)$. By using the Boolean minimization tool *Espresso* [10] we can obtain the following minimal Boolean function representations (BFR) for the four S-boxes in Hummingbird as shown in Table III, where $\overline{x}_i$ denotes the inversion of bit x_i, $\cdot$ denotes a logical AND and $+$ denotes a logical OR. 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 each proposed architecture of the 16-bit block cipher we investigate two implementation strategies (i.e., BFR and LUT) for the four S-boxes in Sections III-C and III-D, respectively, and select one that results in the most area-efficient implementation of the 16-bit block cipher.

[1]Basic building blocks of Xilinx FPGA are CLB slices and each slice on a Spartan-3 device contains two sets of a look-up table (LUT) followed by a flip-flop.

C. Speed Optimized Hardware Architecture

In this subsection we present a speed-optimized hardware architecture for Hummingbird encryption/decryption cores, where the encryption or decryption can be performed with four clock cycles. The main goal of the design is to achieve a high speed and throughput. To this end, we first propose a loop-unrolled architecture for the 16-bit block cipher, followed by a detailed description of the data path architectures of encryption/decryption cores.

1) 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. After the given 16-bit block is XORed with the first round key, the obtained result is split into four 4-bit chunks and each of them is then processed by a 4-bit S-box in parallel. The linear transform L performs a permutation on the output of the S-box layer for each of four regular rounds. The final round only includes the S-box layer and four XOR operations and the output ciphertext is stored into a 16-bit flip-flop (FF).

To select a "hardware-friendly" S-box for the compact version of Hummingbird, we implement the loop-unrolled architecture of the 16-bit block cipher on the target FPGA platform and test one S-box candidate from Table II each time. Table IV summarizes the area requirement when using different S-boxes and implementation strategies. All experimental results are from post-place and route analysis.

TABLE IV
AREA REQUIREMENT COMPARISON FOR THE LOOP-UNROLLED ARCHITECTURE OF 16-BIT BLOCK CIPHER ON THE SPARTAN-3 XC3S200 FPGA (USING FOUR S-BOXES AND TWO IMPLEMENTATION STRATEGIES)

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 IV 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.

2) Speed Optimized Hummingbird Encryption Core: The top-level description and the I/O interface of a speed optimized Hummingbird encryption core are illustrated in Figure 4 and Figure 5, respectively. As can be seen from Figure 5, the speed

TABLE III
BOOLEAN FUNCTION REPRESENTATIONS FOR S-BOXES IN HUMMINGBIRD

S-boxes	Minimal Boolean Function Representations
$S_1(x)$	$S_1^{(0)}(x) = x_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot x_0 + \overline{x}_3 \cdot \overline{x}_2 \cdot x_1 + x_3 \cdot x_2 \cdot x_1 \cdot x_0 + x_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + x_3 \cdot x_2 \cdot x_1 \cdot \overline{x}_0 + \overline{x}_3 \cdot x_1 \cdot x_0$
	$S_1^{(1)}(x) = x_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot x_0 + x_3 \cdot x_2 \cdot x_1 \cdot x_0 + \overline{x}_3 \cdot x_2 \cdot x_1 \cdot \overline{x}_0 + x_3 \cdot \overline{x}_2 \cdot \overline{x}_0 + \overline{x}_3 \cdot \overline{x}_2 \cdot x_0 + x_3 \cdot \overline{x}_1 \cdot \overline{x}_0$
	$S_1^{(2)}(x) = \overline{x}_3 \cdot \overline{x}_2 \cdot x_1 + \overline{x}_2 \cdot x_1 \cdot x_0 + x_3 \cdot x_2 \cdot x_1 \cdot \overline{x}_0 + \overline{x}_3 \cdot \overline{x}_2 \cdot x_0 + x_3 \cdot \overline{x}_1 \cdot \overline{x}_0 + \overline{x}_3 \cdot x_2 \cdot \overline{x}_1 \cdot x_0$
	$S_1^{(3)}(x) = x_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot x_0 + \overline{x}_3 \cdot x_2 \cdot x_1 \cdot \overline{x}_0 + \overline{x}_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + x_3 \cdot x_2 \cdot x_1 \cdot \overline{x}_0 + \overline{x}_3 \cdot x_1 \cdot x_0 + \overline{x}_3 \cdot x_2 \cdot \overline{x}_1 \cdot x_0$
$S_2(x)$	$S_2^{(0)}(x) = \overline{x}_3 \cdot x_2 \cdot \overline{x}_1 \cdot x_0 + x_3 \cdot \overline{x}_2 \cdot x_1 \cdot \overline{x}_0 + x_3 \cdot x_2 \cdot x_1 \cdot x_0 + \overline{x}_3 \cdot \overline{x}_2 \cdot x_0 + x_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + x_3 \cdot \overline{x}_1 \cdot \overline{x}_0$
	$S_2^{(1)}(x) = \overline{x}_3 \cdot \overline{x}_2 \cdot x_1 \cdot \overline{x}_0 + \overline{x}_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot x_0 + \overline{x}_3 \cdot x_2 \cdot x_0 + x_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot x_0 + x_3 \cdot \overline{x}_2 \cdot x_1 \cdot x_0 + x_3 \cdot \overline{x}_1 \cdot \overline{x}_0$
	$S_2^{(2)}(x) = \overline{x}_3 \cdot \overline{x}_2 \cdot x_1 \cdot \overline{x}_0 + \overline{x}_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot x_0 + x_3 \cdot \overline{x}_2 \cdot x_1 \cdot x_0 + x_3 \cdot x_1 \cdot \overline{x}_0 + x_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + x_3 \cdot x_2 \cdot \overline{x}_1$
	$S_2^{(3)}(x) = \overline{x}_3 \cdot x_2 \cdot \overline{x}_1 \cdot x_0 + x_3 \cdot \overline{x}_2 \cdot x_1 \cdot \overline{x}_0 + x_3 \cdot x_2 \cdot x_1 \cdot x_0 + \overline{x}_3 \cdot x_1 \cdot \overline{x}_0 + x_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot x_0 + x_3 \cdot x_2 \cdot \overline{x}_1$
$S_3(x)$	$S_3^{(0)}(x) = x_3 \cdot x_2 \cdot x_1 \cdot \overline{x}_0 + x_2 \cdot \overline{x}_1 \cdot x_0 + x_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + \overline{x}_3 \cdot x_1 \cdot \overline{x}_0 + x_3 \cdot x_2 \cdot x_0 + \overline{x}_3 \cdot \overline{x}_2 \cdot x_1$
	$S_3^{(1)}(x) = x_3 \cdot x_2 \cdot \overline{x}_1 \cdot x_0 + x_3 \cdot x_2 \cdot x_1 \cdot \overline{x}_0 + \overline{x}_3 \cdot x_2 \cdot x_1 \cdot x_0 + \overline{x}_3 \cdot \overline{x}_2 \cdot \overline{x}_1 + x_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + \overline{x}_2 \cdot x_1 \cdot \overline{x}_0$
	$S_3^{(2)}(x) = \overline{x}_3 \cdot x_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + \overline{x}_2 \cdot \overline{x}_1 \cdot x_0 + \overline{x}_2 \cdot x_1 \cdot \overline{x}_0 + x_3 \cdot x_2 \cdot x_0 + \overline{x}_3 \cdot \overline{x}_2 \cdot x_1$
	$S_3^{(3)}(x) = \overline{x}_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot x_0 + \overline{x}_3 \cdot x_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + \overline{x}_3 \cdot x_2 \cdot x_1 \cdot x_0 + x_3 \cdot x_1 \cdot x_0 + x_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + \overline{x}_3 \cdot x_1 \cdot \overline{x}_0$
$S_4(x)$	$S_4^{(0)}(x) = x_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + x_2 \cdot x_1 \cdot x_0 + \overline{x}_3 \cdot \overline{x}_2 \cdot x_1 \cdot \overline{x}_0 + x_3 \cdot \overline{x}_2 \cdot x_1 \cdot x_0 + x_3 \cdot x_2 \cdot x_1 \cdot \overline{x}_0 + \overline{x}_3 \cdot \overline{x}_1 \cdot x_0$
	$S_4^{(1)}(x) = \overline{x}_3 \cdot \overline{x}_2 \cdot x_1 \cdot \overline{x}_0 + x_3 \cdot x_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + x_3 \cdot \overline{x}_2 \cdot x_1 \cdot x_0 + \overline{x}_2 \cdot \overline{x}_1 \cdot x_0 + \overline{x}_3 \cdot x_2 \cdot x_1 + x_3 \cdot \overline{x}_2 \cdot x_1 \cdot \overline{x}_0$
	$S_4^{(2)}(x) = x_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + x_2 \cdot x_1 \cdot x_0 + \overline{x}_3 \cdot x_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + \overline{x}_2 \cdot \overline{x}_1 \cdot x_0 + x_3 \cdot \overline{x}_2 \cdot x_1 \cdot \overline{x}_0 + \overline{x}_3 \cdot \overline{x}_2 \cdot x_0$
	$S_4^{(3)}(x) = x_3 \cdot \overline{x}_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + \overline{x}_3 \cdot x_2 \cdot \overline{x}_1 \cdot \overline{x}_0 + x_3 \cdot \overline{x}_1 \cdot x_0 + x_3 \cdot \overline{x}_2 \cdot x_1 \cdot x_0 + \overline{x}_3 \cdot x_2 \cdot x_1 + x_3 \cdot x_2 \cdot x_1 \cdot \overline{x}_0$

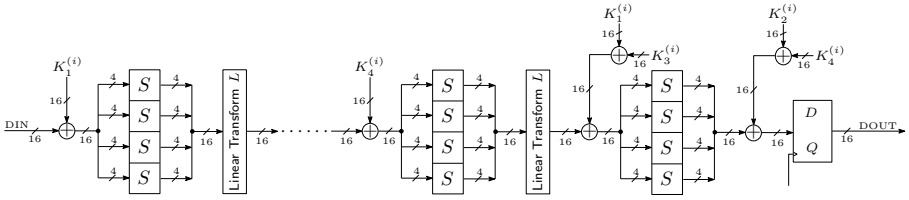


Fig. 3. Loop-Unrolled Architecture of 16-bit Block Cipher

optimized Hummingbird encryption core has 119 pins and therefore can be implemented on the Spartan-3 XC3S200 FPGA that features $200,000$ systems gates and a package (FT256) with 173 I/O pins.

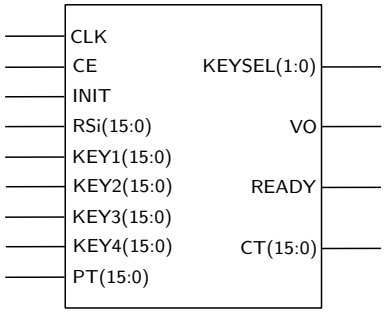


Fig. 5. The I/O Interface of Hummingbird Encryption Core

The speed optimized Hummingbird encryption core works as follows. After the chip enable signal CE 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(15:0) within four clock cycles. From the fifth clock cycle, the core starts encrypting $RS1 \boxplus RS3$ for four times (see Algorithm 1) 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 through the interfaces KEY1(15:0) to KEY4(15:0) and under the control of the signal KEYSEL(1:0). 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. Note that in order to save chip area for the encryption-only core the full update of the rotor $RS2$ involves successive encryptions of two plaintext blocks. More specifically, the rotor $RS2$ is updated by $V12_t$ and $RS4_{t+1}$ (see Algorithm 2) when encrypting two successive plaintext blocks, respectively. Once the initialization process is done after 20 clock cycles, the READY signal changes from '0' to '1' and 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 and the valid output signal VO becomes high level. Therefore, the proposed speed optimized Hummingbird encryption core can encrypt one 16-bit plaintext block per 4 clock cycles, after an initialization process of 20 clock cycles.

3) Speed Optimized Hummingbird Encryption/Decryption Core: We depict the top-level architecture and the I/O interface of a speed optimized Hummingbird encryption/decryption core in the following Figure 6 and Figure 7, respectively. As can be seen from Figure 7, the speed optimized Hummingbird encryption/decryption core has 143 pins and therefore can also be implemented on the Spartan-3

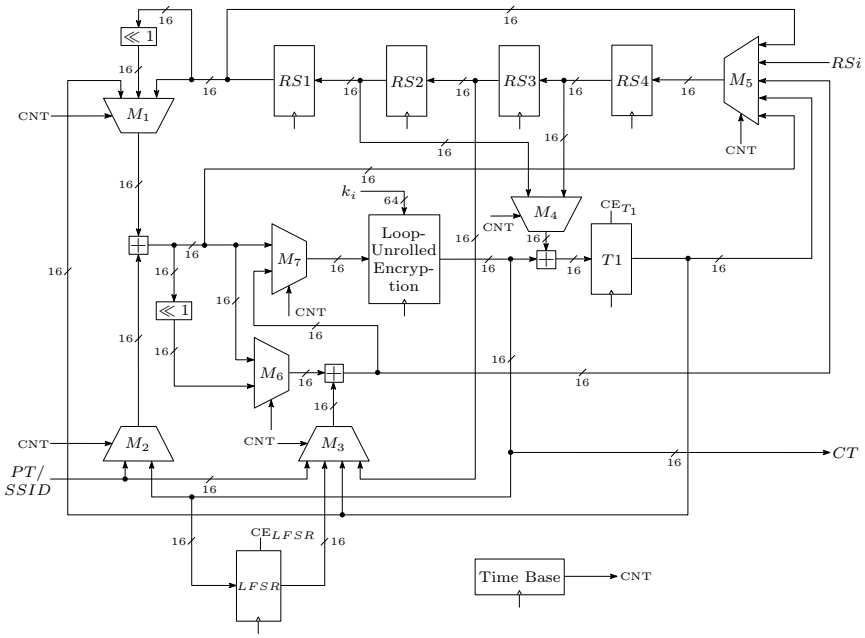


Fig. 4. The Datapath of Speed Optimized Hummingbird Encryption Core Using the Loop-Unrolled Architecture of 16-bit Block Cipher

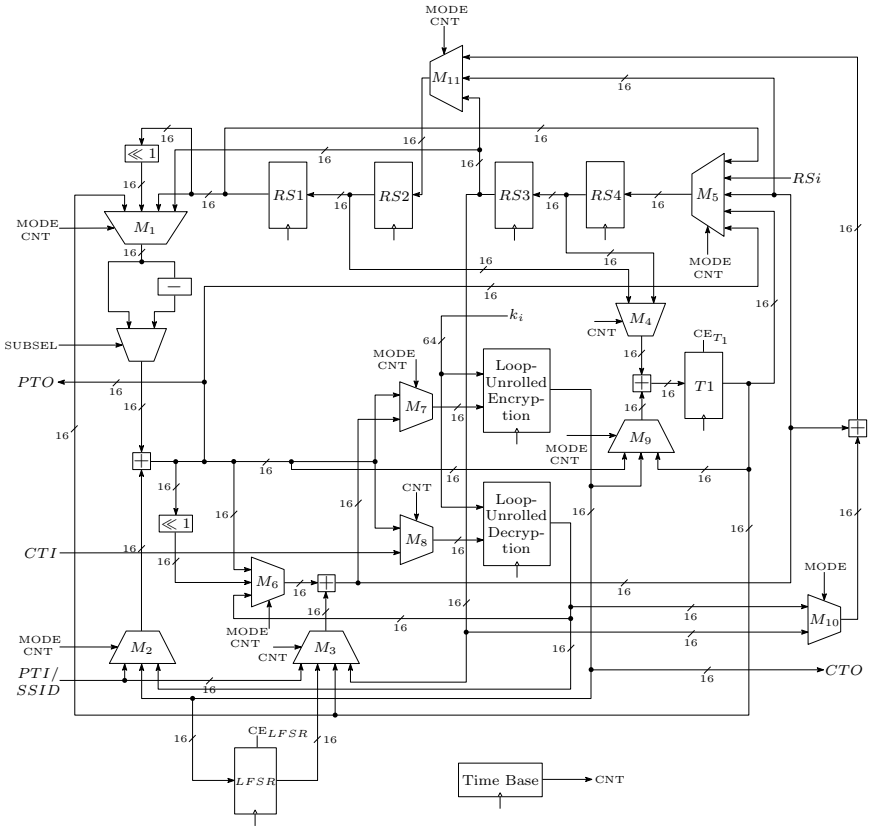


Fig. 6. The Datapath of Speed Optimized Hummingbird Encryption/Decryption Core Using the Loop-Unrolled Architecture of 16-bit Block Cipher

XC3S200 FPGA.

The Hummingbird encryption/decryption core supports the following four operation modes: i) encryption only; ii) decryption only; iii) encryption followed by decryption; and iv) decryption followed by encryption. Both encryption and decryption routines share the same initialization procedure that first takes 4 clock cycles to load four random nonce into rotors through multiplexers M_5 and M_{11}, followed by 16

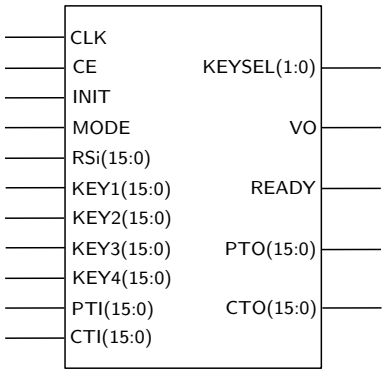


Fig. 7. The I/O Interface of Hummingbird Encryption/Decryption Core

clock cycles for four iterations. The architecture of the encryption/decryption core is quite similar to that of the encryption-only core except the following several aspects. Firstly, the rotor RS_2 completes the update when encrypting two successive plaintext blocks in the encryption-only core, whereas all rotors are fully updated each time a plaintext block is encrypted or decrypted in order to support the four operation modes in the encryption/decryption core. For this purpose, two multiplexers M_{10} and M_{11} are introduced to fully update the rotor $RS2$ after each encryption/decryption. Secondly, an adder that can perform both modulo 2^{16} addition and subtraction is included, which executes the corresponding arithmetic according to the operation modes of the core. Thirdly, two multiplexers M_7 and M_8 are used to feed correct values to the encryption and decryption routines of the 16-bit block cipher, respectively. Finally, all the other multiplexers select appropriate inputs based on the value of a round counter as well as the operation modes. The workflow of the encryption/decryption core is also similar to that of encryption-only core. Hence, the speed optimized Hummingbird encryption/decryption core can encrypt or decrypt one 16-bit plaintext or ciphertext block per 4 clock cycles, after an initialization process of 20 clock cycles.

D. Area Optimized Hardware Architecture

We describe an area-optimized architecture for Hummingbird encryption/decryption cores in this subsection, which require 16 clock cycles to perform the encryption or decryption. Different from the prior speed-optimized design, the area-optimized architecture features a more compact and energy-efficient solution. A round-based architecture for the 16-bit block cipher is first presented. According to the round-based design of the 16-bit block cipher, we devise the corresponding hardware architecture for the area-optimized Hummingbird encryption/decryption cores.

1) Round-based Architecture of 16-bit Block Cipher: To further reduce the chip area and power consumption, we propose a round-based architecture that repeatedly uses only one round function block as shown in Figure 8. In this architecture, four regular rounds share the common hardware resources of one substitution and permutation layer and the final round is composed of another substitution layer and four XORs. Hence, there are totally 5 XORs, 8 S-boxes, and one permutation layer for the datapath. Furthermore, three 16-bit multiplexers are introduced for different purposes: i) a 4-to-1 multiplexer M_1 is utilized to switch among the required round keys; ii) a 2-to-1 multiplexer M_2 is employed to choose between the input and the computation result of each round; and iii) a 2-to-1 multiplexer M_3 is used to export either the computation result of each round or the final ciphertext that is then stored into a 16-bit register. For the round-based architecture, the whole encryption can be performed in four clock cycles.

Similar to the case of the loop-unrolled architecture, we also implement the round-based architecture of the 16-bit block cipher on the Spartan-3 XC3S200 FPGA and test its area requirement when using four S-boxes and two implementation options, respectively. Table V summarizes our experimental results that are from post-place and route analysis on the target platform.

TABLE V
AREA REQUIREMENT COMPARISON FOR THE ROUND-BASED ARCHITECTURE OF 16-BIT BLOCK CIPHER ON THE SPARTAN-3 XC3S200 FPGA (USING FOUR S-BOXES AND TWO IMPLEMENTATION STRATEGIES)

S-box	Implementation Strategy	# LUTs	# FFs	Total Occupied Slices
$S_1(x)$	LUT	158	16	92
	BFR	158	16	85
$S_2(x)$	LUT	156	16	92
	BFR	**152**	**16**	**82**
$S_3(x)$	LUT	154	16	86
	BFR	161	16	92
$S_4(x)$	LUT	159	16	92
	BFR	163	16	93

From Table V we note that the round-based architecture of the 16-bit block cipher can achieve the minimal area on the Spartan-3 XC3S200 FPGA by employing the S-box $S_2(x)$ in each round and implementing it with the boolean function representations. Consequently, the S-box $S_2(x)$ is selected for efficient implementation of area optimized Hummingbird encryption/decryption cores that are addressed in the following subsections.

2) Area Optimized Hummingbird Encryption Core: The top-level description of an area optimized Hummingbird encryption core is depicted in Figure 9. Moreover, the area optimized Hummingbird encryption core has the same I/O interface (see Figure 5) as that of the speed optimized encryption unit.

The area optimized Hummingbird encryption core works as follows. Once the chip is enabled (i.e., CE = '1'), the initialization process (see Figure 1(a)) starts and four rotors RSi $(i = 1, 2, 3, 4)$ are first initialized by four 16-bit random nonce through the interface RSi(15:0) within four clock cycles. Moreover, the value $RS3 \boxplus SSID$ is also stored into the register RA in the fourth clock cycle, where

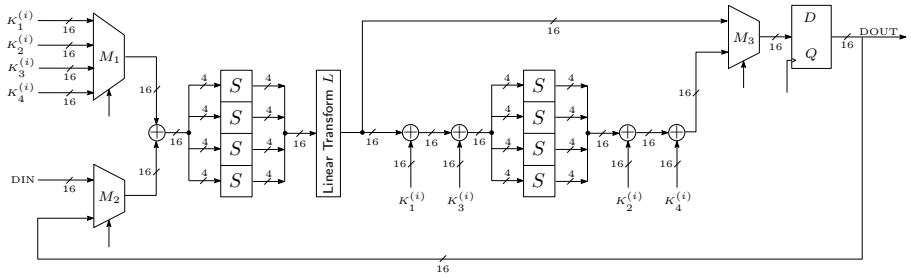


Fig. 8. Round-based Architecture of 16-bit Block Cipher

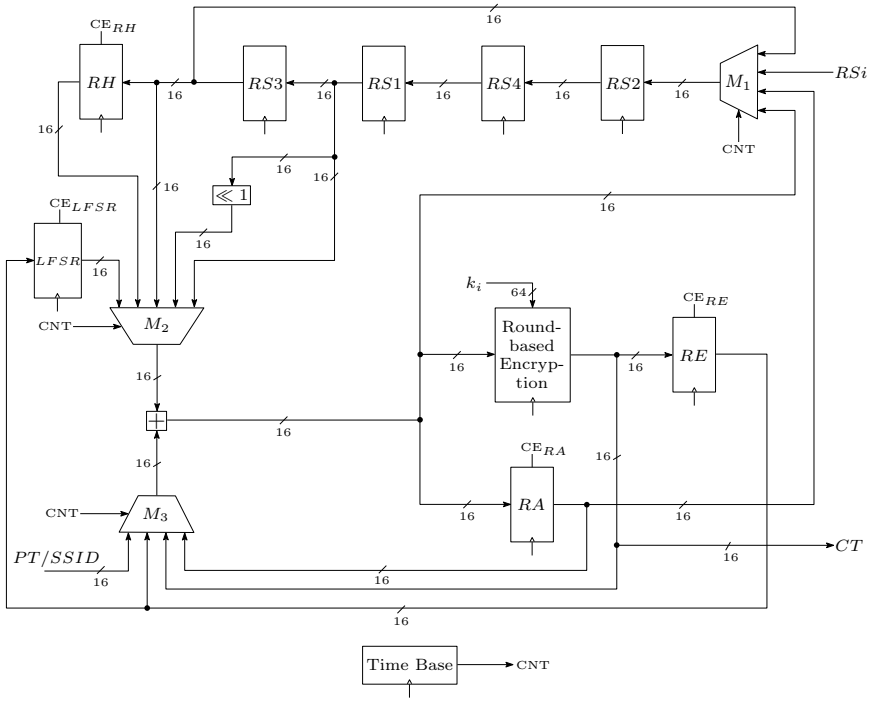


Fig. 9. The Datapath of Area Optimized Hummingbird Encryption Core Using the Round-based Architecture of 16-bit Block Cipher

$SSID$ denotes the identity of current session[2]. The core then executes four iterations (see Algorithm 1) to encrypt the message $RS1 \boxplus RS3$ from the fifth clock cycle. Each iteration takes 20 clock cycles to complete encryptions by four 16-bit block ciphers and the internal state updating as well. Depending on the value of a round counter, the 64-bit subkeys k_i $(i = 1, 2, 3, 4)$ are read from an external register through the interfaces KEY1(15:0) to KEY4(15:0) and under the control of the signal KEYSEL(1:0). In addition, multiplexers M_1, M_2 and M_3 and temporary registers RH, RA and RE also choose the corresponding inputs under the control of the round counter. While the multiplexer M_1 takes care of the update of four rotors, M_2 and M_3 select appropriate operands to form the correct input of the 16-bit block cipher. After 80 clock cycles, the system initialization is finished and the READY signal becomes high level. The first 16-bit plaintext block is then read from an external register for encryption. For another 16 clock cycles, the corresponding ciphertext is output from the encryption core and the valid output signal VO changes from '0' to '1'. Therefore, the proposed area optimized Hummingbird encryption core is able to encrypt one 16-bit plaintext block per 16 clock cycles, after an initialization process of 84 clock cycles.

3) Area Optimized Hummingbird Encryption/Decryption Core: We show the top-level architecture of an area optimized Hummingbird encryption/decryption core in Figure 10. Note that both area and speed optimized encryption/decryption cores have the same I/O interface (see Figure 7).

Like the speed optimized Hummingbird encryption/decryption core, the area optimized one also supports four operation modes (see Section III-C3). Moreover, both encryption and decryption routines share the same initialization procedure that first takes 4 clock cycles to load four random nonce into rotors through multiplexers

[2]Note that session identity $SSID$ is useful when using Hummingbird in some authentication protocols, see [9] for an example. If Hummingbird is only used as an encryption engine, $SSID$ is not necessary and only $RS3$ is stored in RA in the fourth clock cycle.

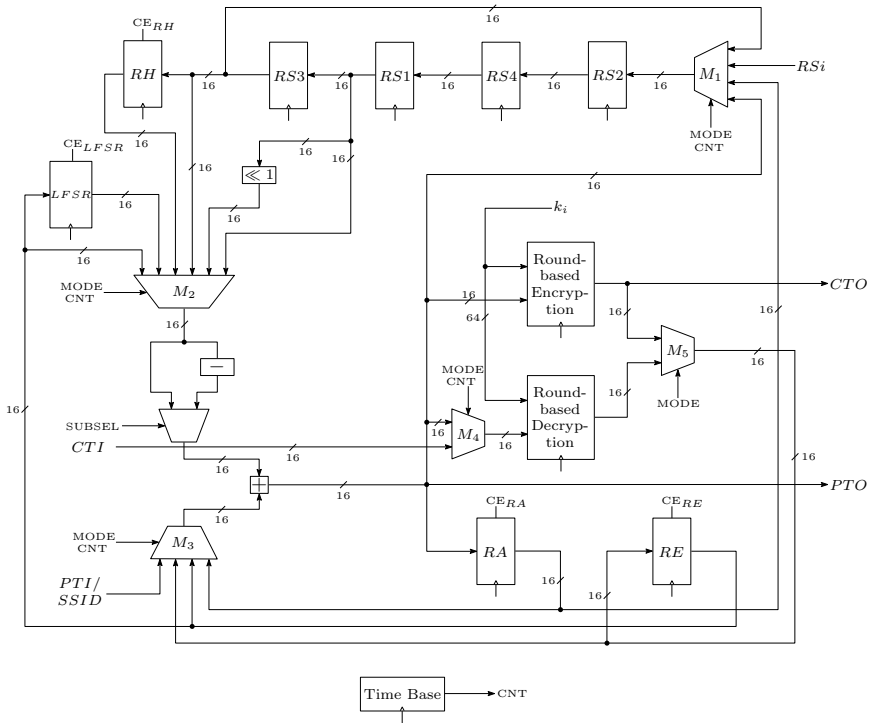


Fig. 10. The Datapath of Area Optimized Hummingbird Encryption/Decryption Core Using the Round-based Architecture of 16-bit Block Cipher

M_1, followed by 80 clock cycles for four iterations. Three temporary registers RH, RA and RE store the required data under the control of the operation mode selection signal MODE and a round counter. In addition, the encryption/decryption core also consists of an 16-bit adder that can perform both modulo 2^{16} addition or substraction. Depending on the current operation mode and the value of the round counter, multiplexers M_2 and M_3 choose appropriate operands that will be used by the 16-bit adder to generate the correct inputs for the 16-bit encryption or decryption module. All rotors will be fully updated through the multiplexer M_1 after each encryption/decryption of a 16-bit plaintext/ciphertext block. While the plaintex and ciphertext block will be read from the interfaces PTI(15:0) and CTI(15:0), the corresponding ciphertext and plaintext will be output through CTO(15:0) and PTO(15:0), respectively. Furthermore, two multiplexers M_4 and M_5 are also included in the encryption/decryption core, where M_4 drives the required inputs to the 16-bit decryption module and M_5 selects output of the 16-bit encryption or decryption module. Since the encryption/decryption core is just a simple extension of the encryption-only core, both of them follow a quite similar workflow. Therefore, the area optimized Hummingbird encryption/decryption core can encrypt or decrypt one 16-bit plaintext or ciphertext block per 16 clock cycles, after an initialization process of 84 clock cycles.

E. 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 v11.1 from Xilinx on a xc3s200-5ft256 Spartan-3 platform with speed grade -5. In addition, to achieve better performance, we set Place & Route Effort Lever (Overall) to be "High" and Place & Route Extra Effort to be "Continue on Impossible".

For the Hummingbird encryption-only and encryption/decryption cores, Table VI shows that the throughput of the speed optimized implementation is 1.42 and 1.1 times larger than that of the area optimized implementation at the cost of additional 20 and 195 slices on the target platform, respectively.

Table VII describes the performance comparison of our Hummingbird implementation with existing FPGA implementations of block ciphers PRESENT [15], [24], XTEA [19], ICEBERG [27], SEA [23] as well as AES [4], [7], [14], [25]. Note that numerous AES hardware architectures, ranging from compact to high speed, have been proposed in literature and we only focus on those implementations using low-cost Spartan 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 comparison 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 VII.

TABLE VI
IMPLEMENTATION RESULTS FOR COMPACT VERSION OF HUMMINGBIRD ON THE SPARTAN-3 XC3S200 FPGA

Opt.	Mode (Enc/Dec)	S-box & its Implementation	# LUTs	# FFs	Total Occupied Slices	Max. Freq. (MHz)	# CLK Cycles		Throughput (Mbps)	Efficiency (Mbps/# Slices)
							Init.	Enc/Dec		
Speed	Enc	$S_3(x)$ with LUT	473	120	273	40.1	20	4	160.4	0.59
	Enc/Dec		1,024	145	558	32.2			128.8	0.23
Area	Enc	$S_2(x)$ with BFR	411	131	253	66.1	84	16	66.1	0.26
	Enc/Dec		651	152	363	61.4			61.4	0.17

TABLE VII
PERFORMANCE COMPARISON OF FPGA IMPLEMENTATIONS OF CRYPTOGRAPHIC ALGORITHMS

Cipher	Key Size	Block Size	FPGA Device	Total Occupied Slices	Max. Freq. (MHz)	Throughput (Mbps)	Efficiency (Mbps/# Slices)
Hummingbird	256	16	Spartan-3 XC3S200-5	273	40.1	160.4	0.59
PRESENT [24]	80	64	Spartan-3 XC3S400-5	176	258	516	2.93
	128	64		202	254	508	2.51
PRESENT [15]	80	64	Spartan-3E XC3S500	271	–	–	–
XTEA [19]	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 [27]	128	64	Virtex-2	631	–	1,016	1.61
SEA [23]	126	126	Virtex-2 XC2V4000	424	145	156	0.368
AES [7]	128	128	Spartan-2 XC2S30-6	522	60	166	0.32
AES [14]			Spartan-3 XC3S2000-5	17,425	196.1	25,107	1.44
			Spartan-2 XC2S15-6	264	67	2.2	0.01
AES [25]			Spartan-2 XC2V40-6	1,214	123	358	0.29
AES [4]			Spartan-3	1,800	150	1700	0.9

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. However, the implementation of the ultra-lightweight block cipher PRESENT is more efficient than that of Hummingbird, although a slightly large (and hence more expensive) device Spartan-3 XC3S400 is required. The main reason is due to the complex internal state updating procedure in Hummingbird cipher. As a result, the control unit is more complicated and the delay of the critical path is much longer in Hummingbird hardware architecture than those in PRESENT core.

IV. CONCLUSION

This paper presented a design space exploration for FPGA implementations of the ultra-lightweight cryptographic algorithm Hummingbird. The proposed speed-optimized and area-optimized Hummingbird encryption/decryption cores can encrypt or decrypt a 16-bit message block with 4 and 16 clock cycles, after an initialization process of 20 and 84 clock cycles, respectively. 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. As the future research, we intend to conduct further cryptanalysis and security evaluations for Hummingbird cipher as well as propose low power ASIC implementations for low-cost RFID tags.

REFERENCES

[1] R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard", available at http://www.cl.cam.ac.uk/~rja14/Papers/serpent.pdf.

[2] S. Babbage and M. Dodd, "The Stream Cipher MICKEY 2.0", ECRYPT Stream Cipher, Available at http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf, 2006.

[3] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher", *The 9th International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2007*, LNCS 4727, P. Paillier and I. Verbauwhede (eds.), Berlin, Germany: Springer-Verlag, pp. 450-466, 2007.

[4] P. Bulens, F.-X. Standaert, J.-J. Quisquater, and P. Pellegrin, "Implementation of the AES-128 on Virtex-5 FPGAs", *Progress in Cryptology - AFRICACRYPT 2008*, LNCS 5023, S. Vaudenay (ed.), Berlin, Germany: Springer-Verlag, pp. 16-26, 2008.

[5] C. De Cannière, O. Dunkelman, and M. Knežević, "KATAN and KTANTAN – A Family of Small and Efficient Hardware-Oriented Block Ciphers", *The 11th International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2009*, LNCS 5747, C. Clavier and K. Gaj (eds.), Berlin, Germany: Springer-Verlag, pp. 272-288, 2009.

[6] C. De Cannière and B. Preneel, "Trivium – A Stream Cipher Construction Inspired by Block Cipher Design Principles", ECRYPT Stream Cipher, Available at http://www.ecrypt.eu.org/stream/papersdir/2006/021.pdf, 2005.

[7] 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, C. D. Walter, Ç. K. Koç, C. Paar (eds.), Berlin, Germany: Springer-Verlag, pp. 319-333, 2003.

[8] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A Survey of Lightweight-Cryptography Implementations", *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 522-533, 2007.

[9] 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, Berlin, Germany: Springer-Verlag, 2010.

[10] N. N. Espresso. Available at http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm, November 1994.

[11] 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.

[12] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong Authentication for RFID Systems Using the AES Algorithm", *The 6th International Workshop on Cryptographic Hardware and Embedded Systems-CHES 2004*, LNCS 3156, M. Joye and J.-J. Quisquater (eds.), Berlin, Germany: Springer-Verlag, pp. 357-370, 2004.

[13] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen, "AES Implementation on a Grain of Sand", *IEE Proceedings Information Security*, vol. 15, no. 1, pp. 13-20, 2005.

[14] 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, J. R. Rao, B. Sunar (eds.), Berlin, Germany: Springer-Verlag, pp. 427-440, 2005.

[15] X. Guo, Z. Chen, and P. Schaumont, "Energy and Performance Evaluation of an FPGA-Based SoC Platform with AES and PRESENT Coprocessors", *Embedded Computer Systems: Architectures, Modeling, and Simulation - SAMOS'2008*, LNCS 5114, M. Berekovic, N. Dimopoulos, and S. Wong (eds.), Berlin, Germany: Springer-Verlag, pp. 106-115, 2008.

[16] P. Hämäläinen, T. Alho, M. Hännikäinen, and T. D. Hämäläinen, "Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core", *The 9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools - DSD 2006*, pp. 577-583, IEEE Computer Society, 2006.

[17] M. Hell, T. Johansson, and W. Meier, "Grain: A Stream Cipher for Constrained Environments", *International Journal of Wireless and Mobile Computing*, vol. 2, no. 1, pp. 86-93, 2007.

[18] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, and S. Chee, "HIGHT: A New Block Cipher Suitable for Low-Resource Device", *The 8th International Workshop on Cryptographic Hardware and Embedded Systems-CHES 2006*, LNCS 4249, L. Goubin and M. Matsui (eds.), Berlin, Germany: Springer-Verlag, pp. 46-59, 2006.

[19] J.-P. Kaps, "Chai-Tea, Cryptographic Hardware Implemenations of xTEA", *The 9th International Conference on Cryptology in India-INDOCRYPT 2008*, LNCS 5356, D.R. Chowdhury, V. Rijmen, and A. Das (eds.), Berlin, Germany: Springer-Verlag, pp. 363-375, 2008.

[20] G. Leander, C. Paar, A. Poschmann, and K. Schramm, "New Lightweight DES Variants", *The 14th Annual Fast Software Encryption Workshop-FSE 2007*, LNCS 4593, A. Biryukov (ed.), Berlin, Germany: Springer-Verlag, pp. 196-210, 2007.

[21] C. Lim and T. Korkishko, "mCrypton - A Lightweight Block Cipher for Security of Low-cost RFID Tags and Sensors", *Workshop on Information Security Applications-WISA 2005*, LNCS 3786, J. Song, T. Kwon, and M. Yung (eds.), Berlin, Germany: Springer-Verlag, pp. 243-258, 2005.

[22] D. Liu, Y. Yang, J. Wang, and H. Min, "A Mutual Authentication Protocol for RFID Using IDEA", *Auto-ID Labs White Paper*, WP-HARDWARE-048, March 2009, available at http://www.autoidlabs.org/uploads/media/AUTOIDLABS-WP-HARDWARE-048.pdf.

[23] 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.

[24] A. Poschmann, "Lightweight Cryptography - Cryptographic Engineering for a Pervasive World", Ph.D. Thesis, Department of Electrical Engineering and Information Sciences, Ruhr-Universitäet Bochum, Bochum, Germany, 2009.

[25] 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 VeryWell Suited for Small Embedded Applications", *International Conference on Information Technology: Coding and Computing - ITCC 2004*, pp. 583-587, 2004.

[26] F.-X. Standaert, G. Piret, N. Gershenfeld, and J.-J. Quisquater, "SEA: A Scalable Encryption Algorithm for Small Embedded Applications", *The 7th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Applications-CARDIS 2006*, LNCS 3928, J. Domingo-Ferrer, J. Posegga, and D. Schreckling (eds.), Berlin, Germany: Springer-Verlag, pp. 222-236, 2006.

[27] 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.

[28] Xilinx Inc., "Spartan-3 FPGA Family Data Sheet", DS099, December 4, 2009, available at http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf.