SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2016; **9**:2200–2216
Published online 14 March 2016 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.1479

RESEARCH ARTICLE

Modeling and optimization of the lightweight HIGHT block cipher design with FPGA implementation

Bassam Jamil Mohd[1]*, Thaier Hayajneh[2], Zaid Abu Khalaf[2] and Khalil Mustafa Ahmad Yousef[1]

[1] Computer Engineering Department, Hashemite University, Zarqa, Jordan
[2] School of Engineering and Computing Sciences, New York Institute of Technology, New York, NY, U.S.A.

ABSTRACT

The growth of low-resource devices has increased rapidly in recent years. Communication in such devices presents two challenges: security and resource limitation. Lightweight ciphers, such as HIGHT cipher, are encryption algorithms targeted for low resource systems. Designing lightweight ciphers in reconfigurable platform (*e.g.*, field-programmable gate array) provides speedup as well as flexibility. The HIGHT cipher consists of simple operations and provides adequate security level. The objective of this research work is to design, optimize, and model FPGA implementation of the HIGHT cipher. Several optimized designs are presented to minimize the required hardware resources and energy including the scalar and pipeline ones. Our analysis shows that the scalar designs have smaller area and power dissipation, whereas the pipeline designs have higher throughput and lower energy. Because of the fact that obtaining the best performance out of any implemented design mainly requires balancing the design area and energy, our experimental results demonstrate that it is possible to obtain such optimal performance using the pipeline design with two and four rounds per stage as well as with the scalar design with one and eight rounds. Comparing the best implementations of pipeline and scalar designs, the scalar design requires 18% less resources and 10% less power, while the pipeline design has 18 times higher throughput and 60% less energy consumption. Copyright © 2016 John Wiley & Sons, Ltd.

KEYWORDS
security; information security; cipher; encryption; cryptography; block cipher; lightweight block cipher; FPGA; power; energy; low-resource devices; HIGHT

***Correspondence**
Bassam Jamil Mohd, Computer Engineering Department, Hashemite University, Zarqa, Jordan.
E-mail: bassam@hu.edu.jo

1. INTRODUCTION

Applications based on smart low-resource devices have rapidly increased in recent years. Examples of such applications include Internet of Things [1], radio frequency identification [2], smart cards [2], wireless body area network [3], and wireless sensor network [2]. For hardware implementations of such applications, many design aspects such as area (or number of gates or logical elements), security, power, and energy need to be considered and balanced in order to obtain the best out of the implemented design. For instance, considering the security aspects of the low resource devices, it is important to provide acceptable security level to protect sensitive and confidential data. However, this might require increasing the area of the design or burden the low resource device when implementing a sophisticated encryption algorithm, which is a fundamental design feature in any security system. Hence, optimized cipher implementations that utilize minimal resources (e.g., area, power, and energy) are essential.

Ciphers targeted for low resource devices are referred to as lightweight ciphers [4]. Lightweight ciphers demand smaller resources compared with conventional ciphers. Consequently, lightweight ciphers have smaller block size (up to 64-bit), simplified key scheduling and simpler transformation functions. While minimizing overall design resources (e.g., area, power, and energy) is the main objective of the lightweight cipher, energy consumption is the most important design aspect [5].

Generally, encryption algorithms are either asymmetric (public-key) or symmetric. Asymmetric ciphers provide better security service compared with symmetric ciphers but require more complex operations and thus may end up with expensive design [6]. As such, in order to obtain an optimized security design implementation for low resource devices, it is usually preferred to use symmetric ciphers. Symmetric ciphers process data either in blocks (known in cryptography as block ciphers) or as stream of bits (known as stream ciphers). Whereas there are many advantages and disadvantages for both block and stream ciphers, block ciphers are generally more common [7]. Typically, block ciphers can be classified into substitution-permutation network, Feistel, stream, and Lai-Massey [8].

Block ciphers in general can be implemented in software or hardware. Hardware implementations are faster and consumes less energy compared with software designs. Hardware designs are implemented by various methods, including field-programmable gate array (FPGA) designs. An FPGA design has the advantages of being flexible and offering a low cost development span. Also, FPGA is an attractive option to implement cipher algorithms because of its reconfigurability feature. Reconfigurability facilitates algorithm agility (switching of ciphers during operation), upload (upgrading to new cipher), and modification [9]. In fact, it has been suggested that the reconfigurability feature of the hardware design will evolve with algorithms and thus will be more immune against new attacks [10].

Being introduced to the hardware FPGA implementation of lightweight ciphers, it is important to highlight one of the well-known lightweight ciphers that is targeted for low resource systems, which is the "HIGHT" cipher. HIGHT is a hardware-oriented lightweight block cipher proposed by Hong *et al.* [11]. The structure of the HIGHT cipher is a variant of the generalized Feistel network [12] and employs simple hardware-oriented transform operations. HIGHT operates on 64-bit block size and 128-bit key length. Hong *et al.* [11] reported that the HIGHT implementation is faster than a low resource Advanced Encryption Standard (AES) implementation. Several articles highlighted the performance of the HIGHT hardware implementation including [5,13–18]. However, none of the studies presented an in-depth analysis of the FPGA designs for the HIGHT cipher, which is the main focus of this article.

The motivation of this work is to examine the impact of the design options on the performance of the lightweight cipher implementation. Consequently, the goal is to optimize and model the performance of the FPGA implemented lightweight cipher. The optimization of the design is meant to utilize minimum hardware resources and reduce consumed power and energy. While other performance metrics are also important (*i.e.*, speed), power and energy are fundamental issues for low-resource devices. The HIGHT cipher is selected because it is intended for low-resource implementation and exhibits good energy and throughput performance [5,13–15]. The reader should notice that the conclusions derived from the HIGHT implementations, presented in this paper, would also apply to other lightweight ciphers. To achieve our aforementioned goal, this research considers the following steps:

- Design and optimize an initial implementation of the HIGHT cipher, which implements one round.
- Examine other implementations with different number of rounds in scalar and pipeline designs.
- Derive models for area, power, and energy to report performance results of the implemented designs.

The rest of the paper is organized as follows. Section 2 presents related work regarding the lightweight ciphers. Section 3 discusses the research flow and methodology. Section 4 briefly describes the HIGHT cipher algorithm. Section 5 presents the scalar implementations. Section 6 discusses the pipeline implementations. Section 7 summarizes the designs' results. Section 8 presents discussion and draws guidelines and trends. Section 9 summarizes concluding remarks.

2. RELATED WORK

Numerous lightweight ciphers were recently published in literature; examples include LEA (2014) [19], Hummingbird2 (2012) [20], Prince (2012) [21], LED (2011) [22], PRINTcipher [23], Hummingbird (2009) [24], KATAN (2009) [13], PRESENT (2007) [7], HIGHT (2006) [11], mCrypton (2005) [25], and Tea/XTea [26,27]. There are several surveys examined software and hardware implementations of the HIGHT cipher and compared them with other ciphers. Following, we just quickly highlight some of those studies.

In software surveys, a set of ciphers is implemented in C or assembly code to measure throughput and memory performance. Eisenbarth *et al.* [14] examined eight block ciphers, targeting embedded software designs. HIGHT ranked second best for throughput. Eisenbarth *et al.* [15] implemented 12 block ciphers in assembly codes, executed on an ATMEL ATtiny45 micro-controller (San Jose, CA, USA), and compared their performance. The HIGHT cipher ranked third in terms of least code size. Additionally, the study stated that the HIGHT cipher provided a good compromise between code size and cycle count. Cazorla *et al.* [8] compared software performance of seventeen lightweight ciphers running on a 16-bit micro-controller. The HIGHT ranked fourth in terms of Random-Access Memory (RAM) requirement.

Other studies examined hardware implementations using application-specific integrated circuit (ASIC) design flow. Eisenbarth *et al.* [14] examined eight block ciphers for embedded designs. The HIGHT cipher ranked the fourth in terms of throughput, and third in cycles per block. Kerckhof *et al.* [5] studied implementations of six ciphers, including the HIGHT cipher. The study showed that the HIGHT cipher had the second best energy per bit performance for encryption/decryption designs. Also, the HIGHT cipher had the least power consumption, and the second least area. Canniere *et al.* [13] examined a set of cipher implementations. It showed that the HIGHT was second in throughput and third in throughput/area.

Lim *et al.* [16] presented an ASIC implementation for the HIGHT encryption and decryption designs. They claimed to have an area reduction of 14% compared with Hong *et al.* [11]. However, no energy analysis was presented. Lee *et al.* [28] presented enhanced ASIC implementation compared with Lim *et al.* [16]. The Lee *et al.* [28] implementation resulted in doubling the throughput and reducing the energy by 38%, but it increased the power and area by about 50%.

Several studies implemented lightweight ciphers in ASIC flows and included HIGHT for comparison, which highlight the significance of the HIGHT cipher. Examples of such studies include Wu *et al.* [29], Badel *et al.* [30], Lee *et al.* [19], Borghoff *et al.* [7], Guo *et al.* [22], Knudsen *et al.* [23]. The studies compared designs implemented in different ASIC process technologies (e.g. 0.25μ and 0.18μ) and, therefore, would not be considered very accurate.

In terms of the FPGA implementations, numerous articles were published to examine several well-known ciphers, such as the AES cipher. For instance, Rahimunnisa *et al.* [31] presented a high throughput FPGA AES implementation; Kundi *et al.* [32] discussed area efficient and low power implementation of the AES. Unfortunately, few articles addressed the FPGA implementation of the HIGHT cipher. Yalla *et al.* [17] discussed the FPGA implementations of the HIGHT and PRESENT ciphers and compared them with published results for AES, Xtea, and Camellia [33]. The study in Yalla *et al.* concluded that the HIGHT implementation had the best throughput over area ratio and the least resources. Koylay *et al.* [18] showed that the HIGHT FPGA implementation had the least area compared with PRESENT, Piccolo [34] and Khudra [35] ciphers. Unfortunately, none of the studies examined different design options to implement the HIGHT cipher in FPGA platform.

To conclude this section, based on the aforementioned cited articles, the HIGHT cipher demonstrated good performance results compared with other ciphers. It has been in several research works to compare against, which highlights its significance. However, the studies in the literature (including the ones earlier) have several shortcomings. Just to list some, first, little research work was invested in the FPGA implementation of HIGHT, specifically exploring different design options. Second, in some cases, the implementations of the ciphers were not optimized thoroughly, which was clearly indicated by the differences in the reported results of the studies. Third, some of the articles compared the performance of the designs of the ciphers implemented under different technologies, which directly effected the results as this naturally introduces significant unavoidable errors. In this paper, many of the aforementioned shortcomings will be dealt with the focus being on studying and optimizing the HIGHT cipher implementations on single FPGA platform using different design options. The presented implementations, optimizations and results could be easily extrapolated and used in the design of other lightweight ciphers.

3. RESEARCH FLOW

To achieve our research goal stated in the Introduction (Section 1), we followed the steps shown in Figure 1. The objective of those steps is to search the space of designs for the optimum implementation of the HIGHT cipher in terms of having the minimum resource usage (*i.e.,* area) and the minimum consumed power and energy. The steps (shown in Figure 1) are as follows:

- Develop the basic design: This is the first step that only considers the scalar (non-pipelined) design (process one block at a time) with only one round of implementation. The concept of "number of implemented rounds" and scalar design will be discussed in Sections 4 and 5, respectively. It is important to emphasize that the main objective of this step is to optimize the basic design components of the HIGHT cipher.
- Develop scalar designs with multiple rounds: This step is very similar to the previous one except that the number of rounds in the scalar design of HIGHT is now being variant (as a power of 2) and being greater than one *i.e.* 2, 4, 8, 16 and 32 rounds. The objective of this step is to optimize the number of implemented rounds.
- Develop pipeline designs with one and multiple rounds. The objective of this step is to optimize the pipeline stages. The pipelined designs will be discussed in Section 6.

In each stage of the design methodology shown in Figure 1, the design must undergo specific design-flow steps, as illustrated in Figure 2. Such flow was used in several previous research work including [36], [37], and [38]. Initially, the design code of the register-transfer level is developed using Verilog$_{TM}$. The code is then validated using dynamic simulations. Next, the code is compiled and synthesized by Altera FPGA Quartus-II$_{TM}$ software package [39], where the targeted FPGA device is cyclone-II. In order to obtain the optimum results, the compilation of the Quartus package is performed twice; the first run is performed with 50-MHz timing constraint, and the second run is performed with the frequency reported from the first run.

The output of the Quartus software mainly generates three reports: timing analysis, resource utilization, and power analysis. Timing analysis reports the maximum

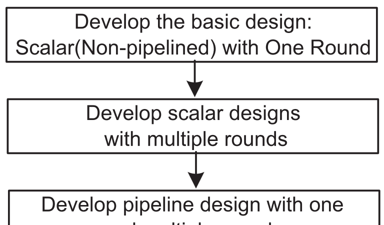

Develop the basic design:
Scalar(Non-pipelined) with One Round
Develop scalar designs
with multiple rounds
Develop pipeline design with one
and multiple rounds

Figure 1. Research flow.

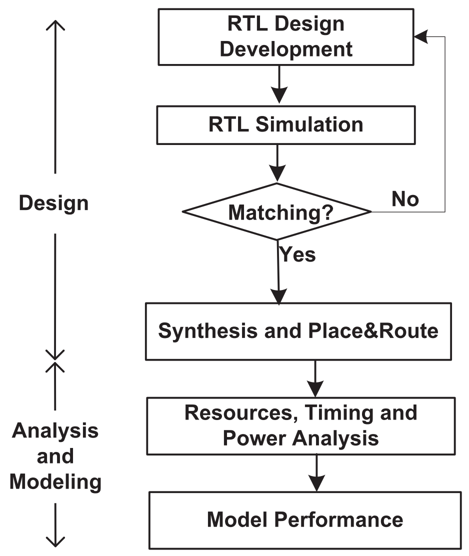


Figure 2. Design flow.

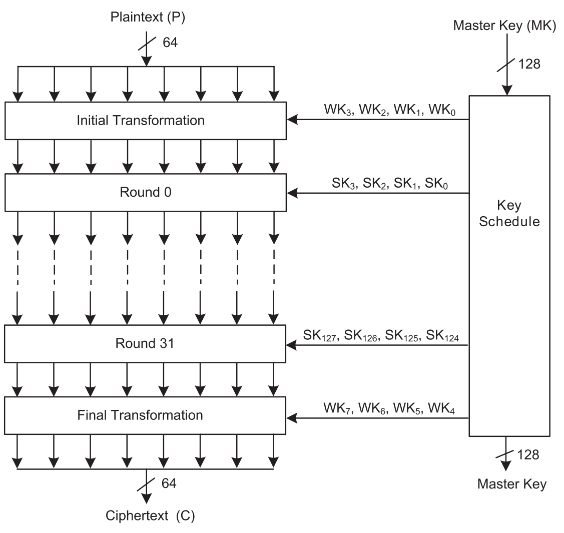


Figure 3. The encryption stage of the HIGHT algorithm.

frequency of the design. The resource utilization report lists the number and type of Logical Elements (LEs) used in the FPGA design. An LE is the smallest building logic in the Altera FPGA (San Jose, CA, USA), and it can be a register, combinational logic, or both [39]. Power analysis computes average power of the design running at 50 MHz based on its circuit activity. The power analysis is similar to the techniques used in [40], [41], and [42]. The analysis requires nodal activities produced by the dynamic simulations using value change dump format. The reported power is basically the core dynamic power, which consists of (i) the combinational logic power, (ii) the clock tree power, and (iii) the register power. Once calculated, the power results are used to estimate the design energy by multiplying the average power by the execution time needed to encrypt one block of data.

The final step in the design flow is to derive models for area, power, and energy. Generally, models are derived mathematically [4] or based on implementation results [37,43]. The algorithm of the HIGHT cipher is described next.

4. THE HIGHT ALGORITHM

The HIGHT cipher is an eight-branch-generalized Feistel network [12]. Figure 3 shows the block diagram of the encryption stage of the HIGHT cipher. The cipher operates on a 64-bit block of plaintext and 128-bit master key (MK) and produces 64-bit cipher text. The encryption stage of HIGHT consists of an initial transformation, 32 rounds of transformations, and a final transformation. Each round of the 32 rounds mainly consists of simple operations: mod 2^8 additions, bitwise rotations, and exclusive-or. The key scheduling consists of two parts: whitening keys (WKs) generation and subkeys (SKs) generation. The WKs are used in the initial and final transformation, where four WKs are used in each transformation. The SKs that used in the 32 rounds of transformation are generated by taking the mod 2^8 addition of the MK and a pseudo-random generated constants. To be specific, 128 SKs are being generated; four SKs are used in each of the 32 rounds. Next, we describe in detail the encryption stage of the HIGHT algorithm using the notations in Table I.

4.1. Key schedule

The key schedule consists of two algorithms for the generation of the keys: WKs and SKs. The WK generating algorithm produces eight WKs for the initial and the final transformations, as shown in Figure 3. The WKs are created directly from the MK as follows:

- $MK_{12}, MK_{13}, MK_{14}$, and MK_{15} are mapped to WK_0, WK_1, WK_2, and WK_3, respectively.
- MK_0, MK_1, MK_2, and MK_3 are mapped to WK_4, WK_5, WK_6, and WK_7, respectively.

The SK-generating algorithm generates the subkeys SK_k for $k = [0, \ldots, 127]$, where subkeys SK_{4i} to SK_{4i+3} are fed to the ith transformation round and the length \of each SK_k is 1 byte. The SK generation employs a primitive polynomial to produce 127 cyclical pseudo-random numbers δ_k for $k = [0, \ldots, 127]$, where $\delta_0 = \delta_{127}$ and δ_0 is set fixed. The polynomial is defined as given in Equation (1) as follows :

$$F(x) = x^7 + x^3 + 1 \qquad (1)$$

The kth pseudo-random number (δ_k) consists of 7 bits $\{s_6, s_5, s_4, s_3, s_2, s_1, s_0\}$ and is generated as given in Equation (2) and discussed further in Section 5.1.

Table I. HIGHT cipher notations

Notations	Description
P_i	The ith byte of the 64-bit input plaintext
C_i	The ith byte of the 64-bit output ciphertext
$X_{i,j}$	The jth byte of the 64-bit of intermediate value input to round i
MK_i	The ith byte of the 128-bit master key (MK)
WK_i	The ith byte of the 64-bit whitening key (WK)
SK_i	The ith byte of the 1024-bit subkey (SK)
δ_i	The ith pseudo-random number generated by the primitive polynomial
'+'	mod 28 addition
$\oplus$	Exclusive-OR operation
$A << n$	n-bit left rotation of byte A

$$\delta_k = \begin{cases} \{1,0,1,1,0,1,0\} & \text{, for } k=0 \\ \{s3_{k-1} \oplus s0_{k-1}, s6_{k-1}, s5_{k-1}, \\ \quad s4_{k-1}, s3_{k-1}, s2_{k-1}, s1_{k-1}\} & \text{, for } k = 1\ldots127 \end{cases} \tag{2}$$

The SK_k is produced using Equation (3) which follows

$$\begin{aligned} SK_{16i+j} &= MK_{((j-i)\ mod\ 8)}\ \text{'+'}\ \delta_{16i+j} \\ SK_{16i+j+8} &= MK_{((j-i)\ mod\ 8)+8}\ \text{'+'}\ \delta_{16i+j+8} \end{aligned} \tag{3}$$

where $0 \le i \le 31, 0 \le j \le 7$.

In the next section, we describe the transformations that are performed during the encryption process of the HIGHT cipher with the help of the generated keys: WKs and SKs.

4.2. The encryption transformations

The initial transformation converts the 64-bit plaintext block P into $X_{0,j}$ bytes or simply X_0, where $j = 0, ..., 7$. In other words, this transformation will generate the j bytes of the 64-bit intermediate value input (X_0) needed in round-0. This is illustrated in Equation (4):

$$\begin{aligned} X_{0,0} &= P_0\ \text{'+'}\ WK_0, & X_{0,1} &= P_1, \\ X_{0,2} &= P_2\ \text{'+'}\ WK_1, & X_{0,3} &= P_3 \\ X_{0,4} &= P_4\ \text{'+'}\ WK_2, & X_{0,5} &= P_5, \\ X_{0,6} &= P_6\ \text{'+'}\ WK_3, & X_{0,7} &= P_7 \end{aligned} \tag{4}$$

The following transformations (not including the final transformation) consist of 32 rounds, where round-i receives intermediate value input X_i and produces X_{i+1}. Each round-i employs two auxiliary functions F_0 and F_1 as given in Equation (5) as follows:

$$\begin{aligned} F_0(X_{i,j}) &= (X_{i,j} \lll 1) \oplus (X_{i,j} \lll 2) \oplus (X_{i,j} \lll 7) \\ F_1(X_{i,j}) &= (X_{i,j} \lll 3) \oplus (X_{i,j} \lll 4) \oplus (X_{i,j} \lll 6) \end{aligned} \tag{5}$$

The generated intermediate value input in each round is given by Equation (6) as follows:

$$\begin{aligned} X_{i+1,0} &= X_{i,7} \oplus (F_0(X_{i,6})\ \text{'+'}\ SK_{4i+3}), & X_{i+1,1} &= X_{i,0}, \\ X_{i+1,2} &= X_{i,1}\ \text{'+'}\ (F_1(X_{i,0}) \oplus SK_{4i+2}), & X_{i+1,3} &= X_{i,2}, \\ X_{i+1,4} &= X_{i,3} \oplus (F_0(X_{i,2})\ \text{'+'}\ SK_{4i+1}), & X_{i+1,5} &= X_{i,4}, \\ X_{i+1,6} &= X_{i,5}\ \text{'+'}\ (F_1(X_{i,4}) \oplus SK_{4i}), & X_{i+1,7} &= X_{i,6} \end{aligned} \tag{6}$$

Now, using the final transformation after the transformations carried in the 32 rounds, the 64-bit cipher text output (C) is generated as shown in Equation (7):

$$\begin{aligned} C_0 &= X_{32,1}\ \text{'+'}\ WK_4, & C_1 &= X_{32,2}, \\ C_2 &= X_{32,3}\ \text{'+'}\ WK_5, & C_3 &= X_{32,4} \\ C_4 &= X_{32,5}\ \text{'+'}\ WK_4, & C_5 &= X_{32,6}, \\ C_6 &= X_{32,7}\ \text{'+'}\ WK_7, & C_7 &= X_{32,0} \end{aligned} \tag{7}$$

It should be noted that the decryption process of the HIGHT cipher is exactly the converse of the encryption process described in this section and thus is not being discussed further.

5. SCALAR DESIGN

This section discusses the FPGA implementation of the scalar design of the HIGHT cipher, which encrypts one block of the plaintext at a time (*i.e.,* the encryption of plaintext blocks is not-pipelined, and it is performed sequentially).

We first show the FPGA basic design implementation of the HIGHT algorithm with one round as presented in Section 4. Then, other FPGA implementations using multiple rounds are discussed and examined to optimize the hardware resources that include but not limited to area, power, and energy of the basic design. Primarily, two aspects of the design optimization are explored: δ-terms generation and number of rounds' implementation. For the former, generating δ terms is one of the main design challenges and as such we examined two implementation methods: linear feedback shift register (LFSR) method [44] and the lookup-table (LUT) method. For the latter, the number of rounds implemented in the hardware is examined for single and multiple rounds.

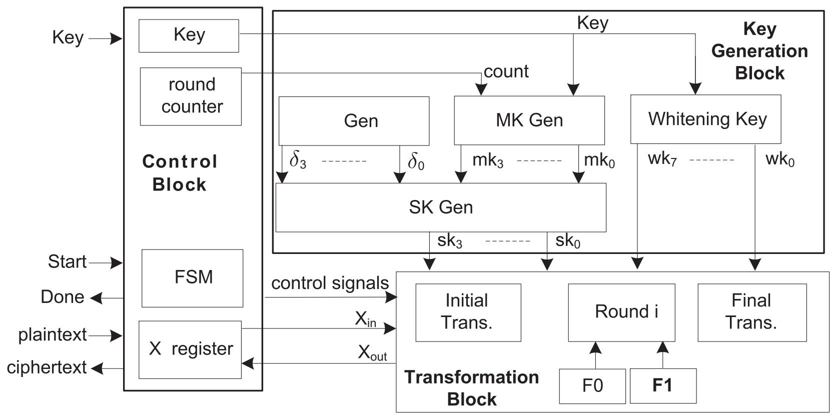


Figure 4. HIGHT FPGA design.

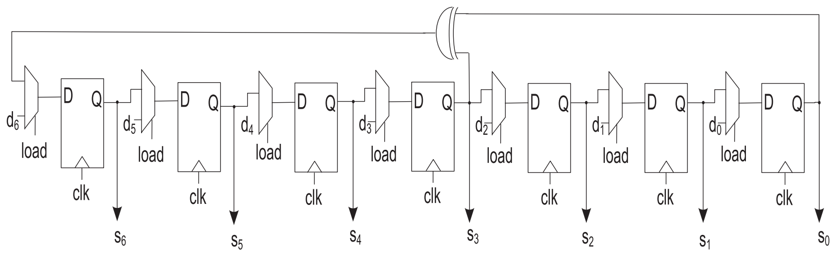


Figure 5. LFSR implementation of the polynomial $x^7 + x^3 + 1$.

5.1. FPGA basic design for one round

The basic design of the one round FPGA implementation of HIGHT is illustrated in Figure 4, which consists of three basic blocks: control logic, key generation, and transformation.

The control block handles the interface with the outside system and performs the sequencing functionality of the activities of the algorithm. The **start** input signal informs the control logic to begin the encryption, while the **done** output signal marks the encryption completion. The control logic manages three registers: key register, round counter, and X-register. The range of the round counter is [0,31], which corresponds to 32 rounds. The finite-state machine responsibility is to organize and sequence the activities of the design.

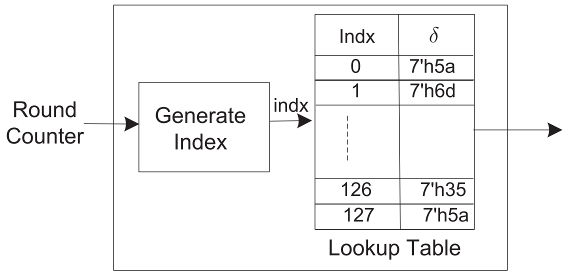
The key generation block produces the WKs and subsequent SKs. The WK is mainly generated by shuffling the MK bytes as was illustrated in Section 4.1. The SKs are generated by combining the δ terms with the MK bytes as was shown in Equation (3).

The transformation block implements the initial transformation, the transformations in the 32 rounds, and the final transformation, as described in Equations (4) to (7).

5.2. The Implementations of δ terms

By algorithm definition, the δ terms are produced by the primitive polynomial in Equation (1). In hardware, primitive polynomials are implemented by LFSR [44]. The LFSR implementing the primitive polynomial in Equation (1) is illustrated in Figure 5. The implementation consists of seven 1-bit registers: $s_0, s_1, ..., s_6$. The registers are connected in a shift-right scheme. On each clock cycle, the registers are fed new values based on the **load** signal being set or reset. If **load** is set, then the registers are loaded the ***d***'s values. If the **load** signal is zero, then (with the exception of s_6) the value from register s_q is fed to register s_{q-1} for $q = 1, \ldots, 6$. Register s_6 is updated with the value $s_3 \oplus s_0$, as specified by Equation (2). Based on the LFSR implementation method, four different δ terms should be produced in each round. Hence, there are four LFSR blocks in each transformation round.

The main disadvantage of the LFSR design is the use of flip-flop circuits (*i.e.,* registers), which are typically bulky and power hungry. One idea to produce δ terms (without flip-flops overhead) is by saving the pre-generated δ values



Indx	δ
0	7'h5a
1	7'h6d
⋮	
126	7'h35
127	7'h5a

Figure 6. δ-Generation by lookup table.

in an LUT as shown in Figure 6. The LUT size is 128 rows $\times$7 bits/row. The index generation logic maps the round counter to LUT index (*i.e.,* address) to access the LUT. Each round employs four LUTs to generate four δ's.

5.3. Results of δ term designs

Table II summarizes the timing analysis and resource utilization results or reports, where these reports already being discussed in Section 3, for both of the LFSR and LUT implementation methods. In terms of timing, the LUT-based design is faster by 10% compared with the LFSR-based design. As for resource utilization, Table II shows the total number of utilized LEs and their types. Both designs seem to use almost the same number of LEs with the LFSR slightly using 1% less. However, the designs differ by the type of utilized LEs. LFSR employs 28 more flip-flops because there are four LFSRs, each with seven flip-flops. The LUT-based design employs more combinational LEs to implement the tables.

Table III summarizes the power and energy results for the δ design implementations. In addition to the total power, Table III breaks down the power into combinational logic power, register power, and clock circuit power. Examining the results reveals that LUT-based implementation consumes less power by 42% compared with the LFSR design. In fact, the LFSR-based design has higher power dissipation in particular in clock and register logic, primarily because of the extra 28 registers. The clock signal has high switching activity compared with other signals. The clock and register circuits are activated more frequently and consume more power.

The energy per block can be computed by multiplying the power by the total time required to encrypt one block. The LUT-based implementation has lower energy by about 42%. Based on the aforementioned analysis, the LUT-based implementation offers superior performance in particular for power and energy consumption. Therefore, we selected the LUT-based design for the rest of our scalar and pipelined implementations of the HIGHT cipher.

5.4. Scalar design with multiple rounds

The single round implementation of the HIGHT encryption, illustrated in Figure 4, requires that the hardware iterates 32 cycles, one cycle for each transformation round-i. As such, the consumed energy (to encrypt a single block) is noticeably high. One approach to reduce the energy of the design is to implement multiple transformation rounds in the hardware as to reduce the number of iterations spent on the ith transformation round and hence reduce the energy dissipated by the flip-flops. The number of iterations for a design that implements n rounds is given by the following equation:

$$\text{Number\_of\_iterations} = \frac{32}{\text{Number\_of\_rounds}} = \frac{32}{n} \quad (8)$$

Figure 7 shows the encryption implementation of the HIGHT cipher with two rounds. The design is slightly modified from the base design shown in Figure 4. The main differences in this design are as follows:

- There are two hardware rounds: Round_{i+0} and Round_{i+1}, which run simultaneously. Hence, the hardware iterates 16 times as opposed to 32 iterations in the basic design (*i.e.,* one round implementation). The range of the round counter is thus [0,. . . ,15], and hence, it employs four bits (*i.e.*, flip-flops) instead of five as found in the basic design.
- In each round, data flow from X-register to Round_{i+0}, then to Round_{i+1}, and finally back to the X-register.
- The key generation generates two sets of SKs, for Round_{i+0} and Round_{i+1}.
- Some blocks do not change between the single round and two round designs, including (most of) the control block, the initial round, final round, and WKs.

The design in Figure 7 can be further scaled up with more rounds (n). We implemented the following number of

Table II. Timing analysis (in terms of maximum frequency) and resource utilization results of δ term designs.

Design	Fmax (MHz)	Resources (# of LEs)			
		Comb.	Reg.	Comb. and Reg.	Total
LFSR-based design	60.72	396	77	152	625
LUT-based design	66.77	431	83	118	632

Table III. Summary of power and energy results of δ term designs.

	Power (mW)				Energy (pJ/block)
	Comb.	Reg.	clk	Total	
L FSR-based Imp.	4.00	2.64	6.67	13.31	9583
L UT-based Imp.	3.55	2.16	3.64	9.35	6732

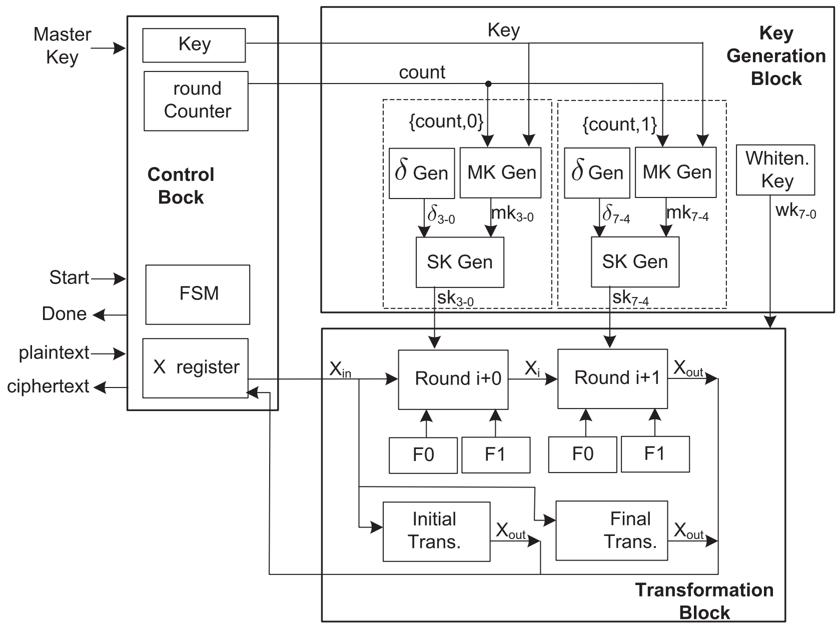


Figure 7. HIGHT FPGA design with two rounds.

rounds: 2, 4, 8, 16, and 32. We compiled and analyzed each design. The results of the hardware performance metrics are summarized in Section 7.1.

6. PIPELINED DESIGN

Compared with the scalar design, the pipelined design is capable of processing more than one task at a time and thus is expected to have better throughput. The pipelined version of the HIGHT cipher encryption design is illustrated in Figure 8(a). It consists of 32 stages; each stage implements one transformation round. Because the initial and final transformations are tiny, they are included in the first and last stages, respectively. At each clock cycle, a new plaintext block is fed to the first stage, processed, and passed along to the next stage. At the end of each stage, the data is saved in the intermediate registers.

The MK moves down the pipeline stages with the processed data. The assumption in Figure 8(a) is that the MK changes every clock cycle. This design is referred to as *key-pipelined design*. In low-resource devices (*e.g.,* wireless sensor network), the cipher key is updated infrequently, and therefore, the key expansion algorithm is executed infrequently [36]. As such, another option to implement the pipeline is to capture the key in a register, which feeds the stages as illustrated in Figure 8(b). This design is referred to as *no-key-pipelined design*. The MK is assumed to be stable for considerable number of cipher blocks. To update the MK, the pipeline completes its current encryptions; then new key is loaded, and encryptions are resumed.

The performance metrics of the two MK pipelined options are summarized in Tables IV and V. Table IV shows that the *key-pipelined design* utilizes 44% extra resources compared with the *no-key-pipelined design*. This percentage of extra resources is mostly due to the used key registers in the pipelined stages. In terms of timing results, the two designs have almost the same maximum operating frequency.

In terms of power consumption, both designs dissipate almost the same total dynamic power and thus the same amount of energy to encrypt one block. Just to elaborate on the latter result in Table V and the energy calculation, the *key-pipelined design* dissipates more power for the register and clock categories, because it employs more registers. The *no-key-pipelined* design has higher combinational power due to higher interconnect power. The interconnect power, which is part of the combinational power, is dissipated while switching the connecting wires. The *no-key-pipelined* design has lengthy wires because it moves the key down the pipeline. For the energy calculation, if we assume having k blocks, then it takes $(k + 31)$ clock cycles to encrypt the k blocks. Thus, the number of clock cycles to encrypt one block (CC) can be computed by Equation (9):

$$\mathrm{CC} = \frac{k+31}{k} \tag{9}$$

for $k \gg 31$ then $\mathrm{CC} \approx 1$.

The computed energy consumption (listed in Table V) shows that both designs dissipates almost the same energy to encrypt one block.

In summary, both pipelined designs earlier have similar performance metrics, with the exception of the resource utilization. The *no-key pipeline* utilizes fewer resources.

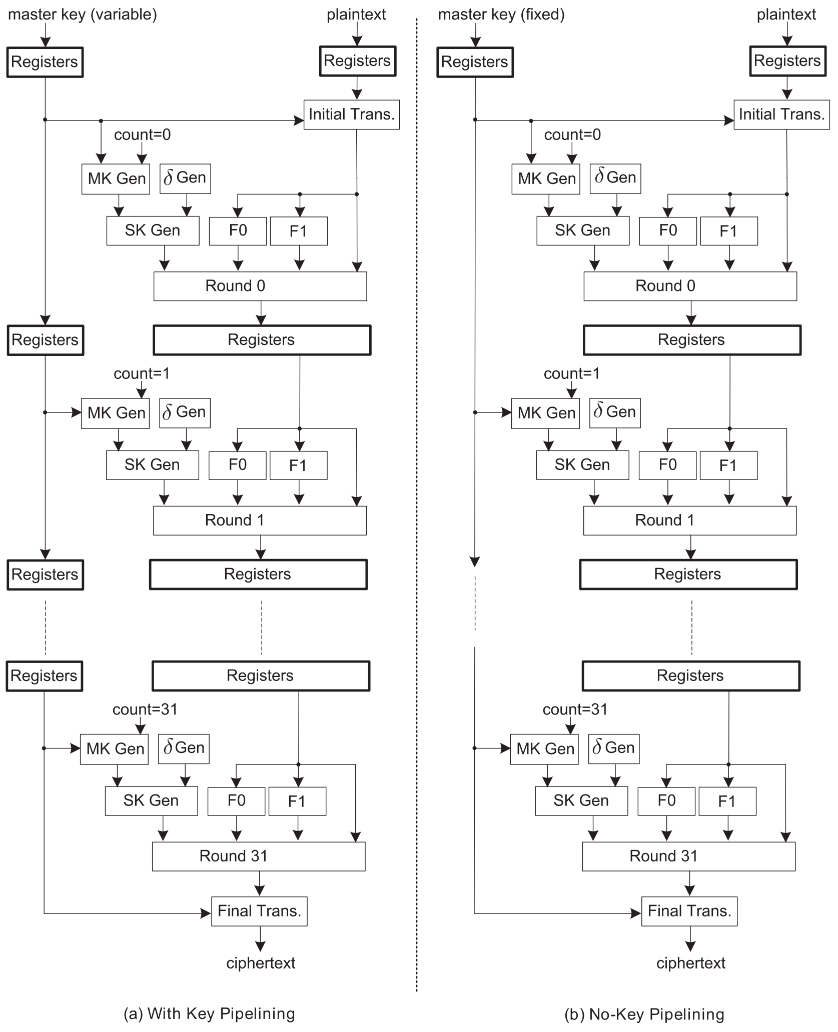


Figure 8. The pipelined design of the HIGHT cipher (encryption stage).

Table IV. Timing analysis (in terms of maximum frequency) and resource utilization results of δ term designs.

Design	Fmax (MHz)	Resources (# of LEs)			
		Comb.	Reg	Comb and Reg	Total
Key Pipelining	67.93	1727	2715	2017	6459
No Key Pipeline	68.15	2307	957	1229	4493

Table V. Summary of power and energy results of the *key-pipelined* and *no-key pipelined* designs.

	Power (mW)				Energy (pJ/block)
	Comb.	Reg	clk	Total	
Key pipelining	21.62	62.68	12.0	96.30	1926
No key pipelining	42.06	44.18	10.93	97.17	1943

Furthermore, the *non-key-pipeline* is closer to the low-resource updating profile as it assumes infrequent updating. Hence, the *no-key-pipeline* was selected for reporting the results in the next section and will be referred to as "the pipeline design" for simplicity.

7. IMPLEMENTATION RESULTS

The results of scalar and pipeline implementations are summarized in this section. As elaborated in Section 3, the presented results consist of the following metrics: the maximum frequency, the number of LEs, the power, and the energy consumption. The following notations will be used throughout this section:

- H_n indicates the FPGA scalar implementation, where the subscript n indicates the number of rounds being implemented in hardware; $n \in 1, 2, 4, 8, 16, 32$.
- HP_n indicates the FPGA pipelined implementation, where the subscript n indicates the number of rounds being implemented in hardware per stage. Consequently, the number of pipeline stages (S) is set to be equal to $32/n$. For instance, HP_2 has 16 stages, each with two transformation rounds.

The results reported in this section are in two forms: Tabular and Graphical. The tables show the raw data to illustrate the absolute values and the power components. The graphical plots present a visual form to show the trend of the computed metrics. The metrics in the plots are normalized with respect to H_1. The tabular results and graphics are both used to derive the performance metric models for the frequency, area, power, and energy.

7.1. Scalar implementations

This section provides a summary of the FPGA scalar implementations that were discussed in Section 5, along with the computed metrics results. The computed metrics are summarized in Table VI and graphically illustrated in Figures 9–14.

Figure 9 shows the trend curve of the maximum frequency with the number of rounds for the scalar implementations. Evidently, doubling the number of rounds reduces the frequency by an average of 12 MHz or equivalently by an average of 18% of H_1 frequency. In fact, the maximum frequency of H_{32} is 10% of that of H_1. The drop of frequency is justified by increasing the length of the timing paths when adding more rounds in one cycle. From Figure 9, the frequency (f) of $H_n = H_{2k}$ for $k = [1, 2, 4, 8, 16]$ can be modeled using Equation (10) with an average model error of 9.3%:

$$f_{H_{2k}} = f_{H_k} - 0.18 \times f_{H_1}, \text{ where: } f_{H_1} = 67\text{MHz} \qquad (10)$$

Figure 10 illustrates the trend curve of the number of LEs in scalar implementations with the number of rounds. The curve indicates that doubling the number of rounds increases the number of resources by 44%. To understand the source of this increasing amount, Figure 11 illustrates the number of LEs for the registers (*i.e.,* dedicated logic registers) and the combinational LEs. Clearly, the number of register LEs is constant, which is expected. On the other

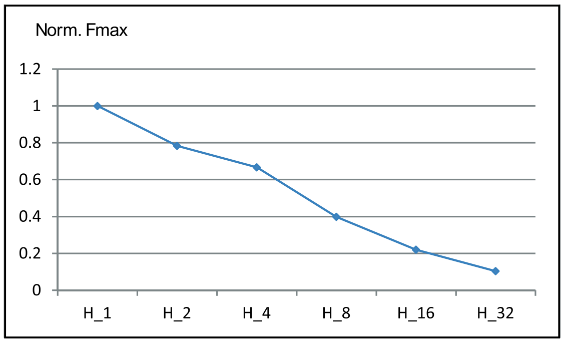


Figure 9. Frequency trend for scalar implementations.

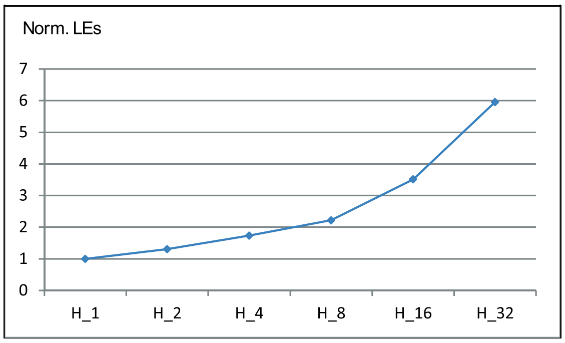


Figure 10. Resource utilization trend for scalar implementations.

Table VI. Summary of the scalar implementations results.

Scalar implementation	F_{max} (MHz)	LEs	Power (mW)	Energy (pJ/block)
H_1	66.77	632	9.35	6732
H_2	52.32	826	18.51	7404
H_4	44.55	1097	22.94	5505.6
H_8	26.61	1402	27.88	4460.8
H_{16}	14.71	2,222	34.46	4135.2
H_{32}	6.94	3765	30.19	3019

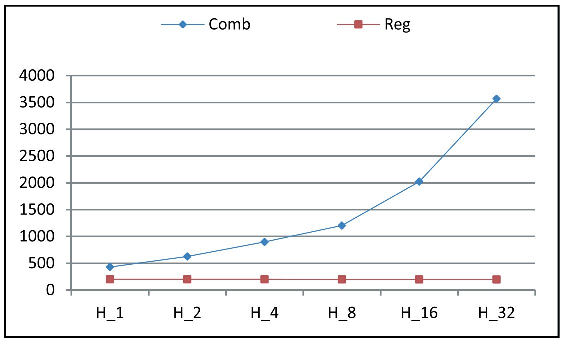


Figure 11. Number of combinational and register LEs for scalar implementations.

hand, the number of combinational LEs grows by 50%, when doubling the number of rounds as the synthesis tool efficiently minimizes and shares the logic cones. From the previous figures, we can model the trend of the number of LEs of $H_n = H_{2k}$ using Equation (11) with an average model error of 6.5%.

$$\begin{aligned} LE_{H_{2k}} &= LE_{H_{2k}^{Comb}} + LE_{H_1^{Register}} \\ &= 1.5 \times LE_{H_k^{Comb}} + LE_{H_1^{Register}} \end{aligned} \qquad (11)$$

where: $LE_{H_1^{Comb}} = 430$ and $LE_{H_1^{Register}} = 200$

Figure 12 depicts the power trend curve. Generally, increasing the number of rounds increases power. The exception is with the 32 rounds design for a reason to be explained next.

Figure 13 illustrates the contribution of combinational logic and sequential logic (*i.e.,* register and clock) to the total power. Doubling the number of rounds increases the sequential power by 12% and the combinational power by 55%. The increase in combinational power is due to the glitch power as described in [4]. The main reason for the drop in the power trend at H_{32} is due to the combinational power, which is mainly higher for H_{16} due to the routing power. Such anomaly is related more to how the software tool places LEs and routes the connections. Clearly, in the case of H_{32} the optimization algorithm worked better to

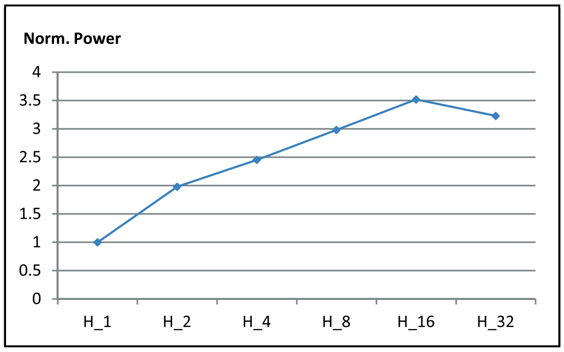


Figure 12. Power trend for scalar implementations.

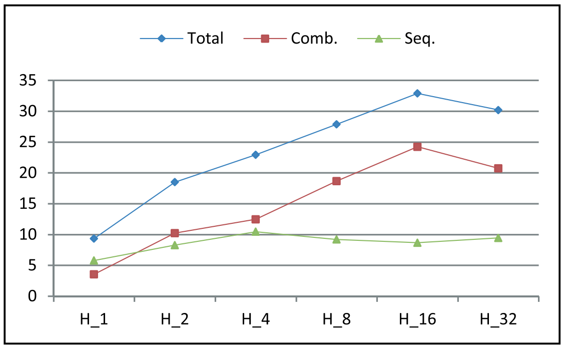


Figure 13. The power components versus number of rounds.

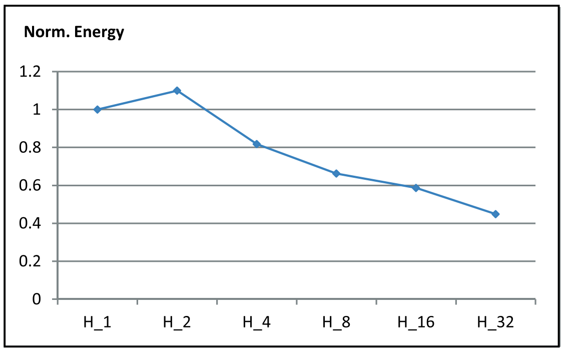


Figure 14. Energy trend for scalar implementations.

reduce the connections. Overall, the trend of the power (P) can be modeled using Equation (12) with an average error of 22.2%.

$$\begin{aligned} P_{H_{2k}} &= P_{H_{2k}^{Comb}} + P_{H_{2k}^{Seq}} \\ &= 1.55 \times P_{H_k^{Comb}} + 1.12 \times P_{H_k^{Seq}} \end{aligned} \qquad (12)$$

where: $P_{H_1^{Comb}} = 4$ mW, and $P_{H_1^{Seq}} = 6$ mW

Finally, Figure 14 shows the energy trend curve. When doubling the number of rounds, the energy drops almost linearly by roughly about 14%. H_2 implementation has the highest energy dissipation. Energy is computed by multiplying average power with time to encrypt a block, which is dependent on the number of cycles to encrypt a block. It is further observed that increasing the number of rounds (n) increases power and reduces time to encrypt a block. Such behavior creates a maximum value at H_2.

H_{32} implementation has the least energy dissipation. This is mainly due to the saving in clock and register powers, because they become more inactive with higher rounds as demonstrated in Figure 15. The trend of the energy (E) can be modeled and estimated using Equation (13) that follows, with an average error of 5.7%.

$$\begin{aligned} E_{H_{2k}} &= E_{H_k} - 850 \text{ pJ} \\ \text{where: } E_{H_1} &= 7350 \text{ pJ} \end{aligned} \qquad (13)$$

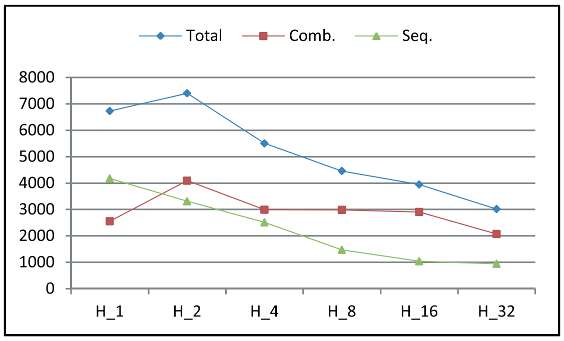


Figure 15. Energy components for scalar implementations.

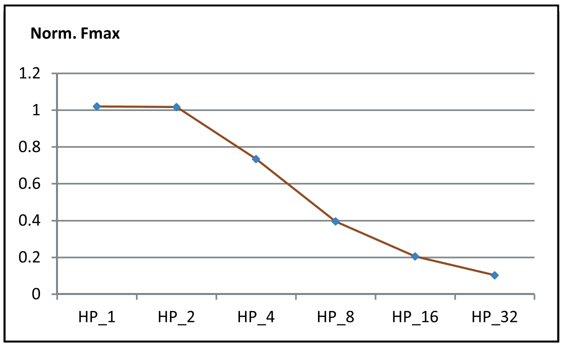


Figure 16. The frequency trend versus the number of rounds per stage for the pipelined FPGA implementations.

7.2. Pipelined implementations

This section presents the results of the FPGA pipelined implementations that was discussed in Section 6. The computed metrics are summarized in Table VII and graphically illustrated in Figures 16 to 21.

The maximum frequency trend with the number of rounds (in pipeline stage) is illustrated in Figure 16. It is noticed that doubling the number of rounds per stage reduces the frequency by about 12.3 MHz, which is close to the non-pipelined or scalar implementation. Also, the increase in the timing paths as more rounds are added to each pipelined stage slows down the frequency. The trend of the frequency of $HP_n = HP_{2k}$ for $k = [1, 2, 4, 8, 16]$ is modeled using Equation (14) with an average error of 14.9%.

$$\begin{aligned} f_{HP_{2k}} &= f_{HP_k} - 0.18 \times f_{HP_1} \\ \text{where: } f_{HP_1} &= 67\text{MHz} \end{aligned} \tag{14}$$

The number of LEs trend curve with the number of rounds per stage is shown in Figure 17, which has a dish-like shape. With the exception of HP_1 and HP_{32}, the resource utilization is almost the same across the different rounds per stage. To better understand the LE trend curve, Figure 18 plots the number of utilized LEs for the registers and the combinational-only LEs. In the plot, the register LEs include register-only and register-and-combinational LEs. HP_1 has the largest number of registers, because it pipelines the partial results every round. It is important to notice that as the number of rounds is doubled per stage, the register LEs decay by 34%, and the combinational LEs grow by 9%, which explains the dish-like curve in Figure 17. Evidently, the number of LEs for HP_{2k} can be modeled as in Equation (15). The average error of this model is 2.3%.

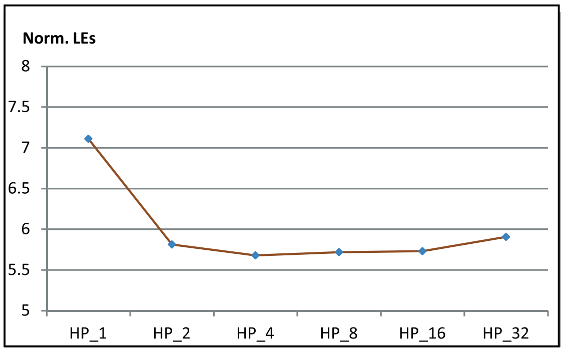


Figure 17. The resource utilization trend versus number of rounds per stage for the pipelined FPGA implementations.

$$\begin{aligned} LE_{HP_{2k}} &= LE_{HP_{2k}^{Comb}} + LE_{HP_{2k}^{Register}} \\ &= 1.09 \times LE_{HP_k^{Comb}} + 0.66 \times LE_{HP_k^{Register}} \\ \text{where: } LE_{HP_1^{Comb}} &= 2307 \text{ and } LE_{HP_1^{Register}} = 2186 \end{aligned} \tag{15}$$

Figure 19 illustrates the power trend curve across the pipelined implementations. The power curve initially decreases to minimum at HP_2 and then picks up gradually from HP_4 to HP_{32}. The justification for this trend of the curve is shown in Figure 20, which plots the curves for total power, combinational power, and sequential logic power. As the number of the registers decreases, the sequential circuit power decays by 30%, while the combinational power

Table VII. Summary of the pipelined implementations results.

Pipeline implementation	F_{max} (MHz)	LEs	Power (mW)	Energy (pJ/block)
HP_1	68.15	4493	97.17	1943.4
HP_2	67.92	3673	90.56	1811.2
HP_4	49.06	3589	95.72	1914.4
HP_8	26.43	3613	127.24	2544.8
HP_{16}	13.73	3621	186.62	3732.4
HP_{32}	6.87	3732	235.74	4714.8

grows by 40%. From Figures 19 and 20, the trend of the power curve for HP_{2k} can be expressed as in Equation (16)

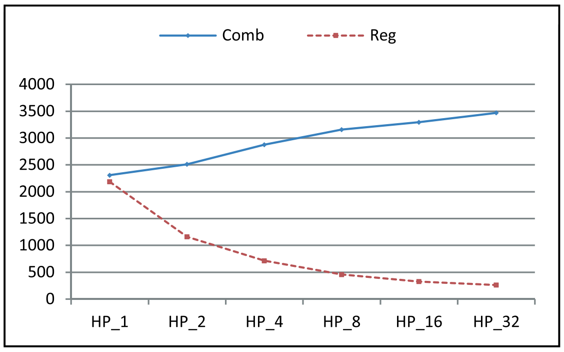


Figure 18. The number of register and combinational LEs versus number of rounds per stage for the pipelined FPGA implementations.

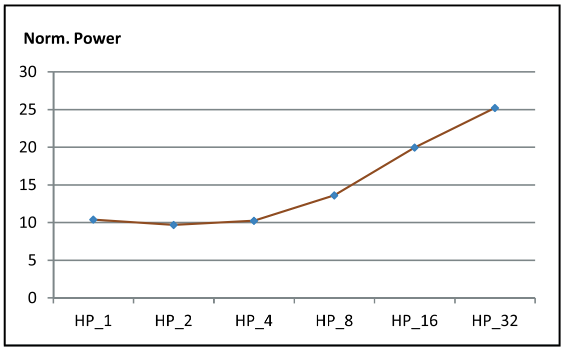


Figure 19. The power trend versus number of rounds per stage for the pipelined FPGA implementations.

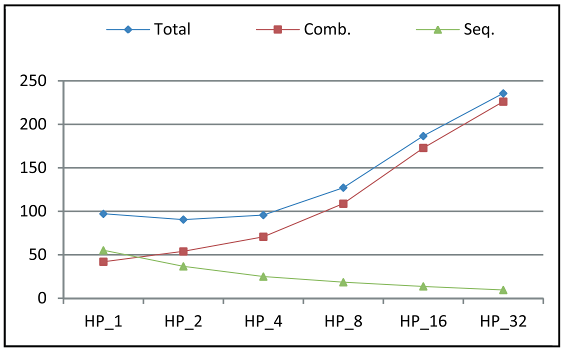


Figure 20. The power components versus number of rounds per stage for the pipelined FPGA implementations.

with an average error of 5.7%.

$$\begin{aligned} P_{HP_{2k}} &= P_{HP_{2k}^{Comb}} + P_{HP_{2k}^{Seq}} \\ &= 1.4 \times P_{HP_k^{Comb}} + 0.7 \times P_{HP_k^{Seq}} \quad (16) \end{aligned}$$

where: $P_{HP_1^{Comb}} = 44$ mW, and $P_{HP_1^{Seq}} = 55$ mW

In Figure 20, the decay in the sequential power is almost the same as the decay in the number of register LEs. However, the growth in combinational power (*i.e.,* 1.4) is larger than the growth of combinational LEs (*i.e.* 1.09). This suggests that the increase in power is attributed to other factors other than the increase in combinational LEs. The other contributing factor might be related to the interconnect power and glitch power [4].

The energy trend is displayed in Figure 21. For pipeline design, energy is computed by multiplying average power with the cycle time. This is because every cycle produces one encrypted block, assuming the pipeline is full. The energy curve in Figure 21 has almost the same dish-like shape (similar to the power curve) with minimum energy at HP_2. Figure 22 illustrates the energy components versus the number of rounds per stage. Doubling the number of rounds results in 40% growth in the combinational energy and in a decay by 30% of the sequential energy.

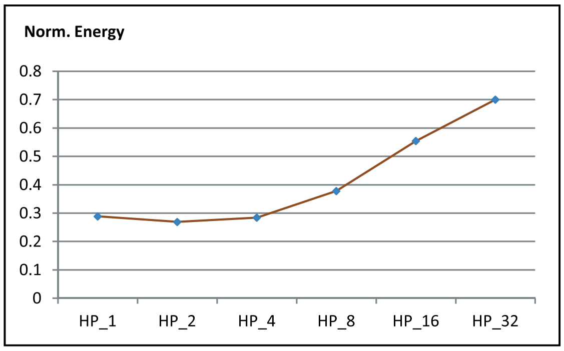


Figure 21. The energy trend versus number of rounds in pipeline stage for the pipelined FPGA implementations.

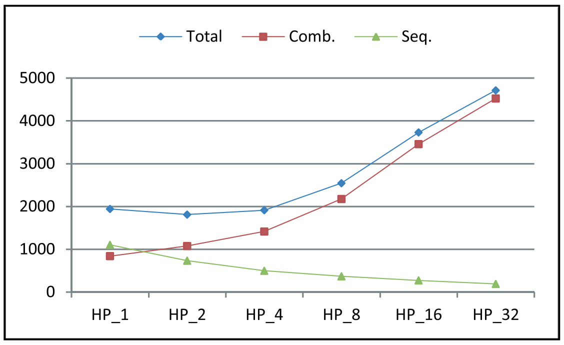


Figure 22. The energy components versus number of rounds per stage for the pipelined FPGA implementations.

The growth in combination energy can be explained by the interconnect and glitch powers, whereas the decay in sequential power is related to the reduction in the number of flip-flops. The trend of the energy of HP_{2k} can be estimated using Equation (17) with an average error of 5.7%.

$$\begin{aligned} E_{HP_{2k}} &= E_{HP_{2k}^{Comb}} + E_{HP_{2k}^{Seq}} \\ &= 1.4 \times E_{HP_k^{Comb}} + 0.7 \times E_{HP_k^{Comb}} \quad (17) \end{aligned}$$

where: $E_{HP_1^{Comb}} = 840 \text{ pJ}, E_{HP_1^{Seq}} = 1100 \text{ pJ}$

8. DISCUSSION

In this section, the scalar and pipelined results presented in Section 7 are discussed to draw guidelines and conclusions. The timing/speed and throughput metrics are discussed first. This is followed by discussing the power and utilized resources (LEs) metrics. Then followed by considering the energy metric. Finally, this section is concluded by highlighting what implementation should constitute the optimum design.

8.1. Speed and throughput

The maximum frequency tends to vary similarly in the scalar and pipeline implementations. The fastest pipelined and scalar designs were HP_1, HP_2, and H_1. All of these designs had fewer number of rounds to execute in one clock cycle. In Figure 23, we show the throughput measured from all of the implemented scalar and pipelined designs in million blocks per second. When comparing the best implementations of both, it can be noticed that the pipelined implementations have considerably higher throughput (about 18 multiples compared with the scalar ones). Within the pipeline implementations, the best implementations were HP_1 and HP_2. For the scalar designs, the best implementations were H_4 followed by H_8.

8.2. Power and LEs

The trends of the power and the number of LEs seem to correlate in most of the implementations. Doubling the number of rounds implemented (in the hardware) in the *scalar* designs results in the following:

- 50% growth in combinational LEs and 55% growth in combinational power; and
- almost no change in sequential LEs and a slight increase in sequential power.

Whereas doubling the number of transformation rounds in each pipeline-stage in the *pipelined* designs results in the following:

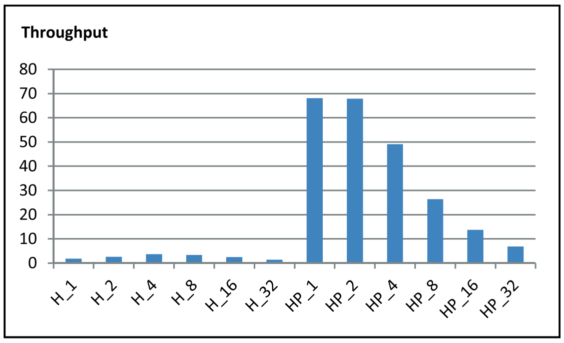


Figure 23. Throughput (million blocks per second).

- 9% growth in combinational LEs and 40% growth in combinational power; and
- 34% decay in sequential LEs and 30% decay in sequential power.

The correlation between the dynamic power and the number of LEs is justified because the dynamic power is proportional to design area, which is proportional to the number of LEs. Also, the implementations perform the same task (HIGHT cipher encryption steps). Being that said, there are still slight differences between power and area trends, which are mostly due to interconnect and glitch powers as well as synthesis tool issues.

Drawing some conclusions from the observations earlier, the scalar designs have in general smaller number of LEs and consume less power compared with the pipelined ones. When comparing the best implementations of both, the number of LEs in scalar design is 18% of the number of LEs of that of the pipeline designs, and the scalar power is 10% of the pipeline power. As of the best implementation, H_1 has the least power and least number of LEs.

8.3. Energy

Figure 24 illustrates the energy trend curve (in petajoule) across all of the implemented scalar and pipelined designs. The curve suggests a continuous spectrum across the x-axis with easily identified minimum and maximum energy values. Pipelined designs generally have lower energy. The lowest energy of pipelined designs is HP_2 closely followed by both HP1 and HP_4. For scalar designs, H_{32} had the lowest energy.

The reason for having a V-like shape curve in Figure 24 is due to the different energy trends for combinational and sequential logic. The sequential energy tends to decrease from H_1 to HP_{32}. The combinational power also tends to decrease in H_n with the increments of rounds in scalar designs. On the other hand, it grows with HP_n with the increase in the number of rounds in the stages of the pipelined designs. The combined results create the V-shape of the energy plot. Comparing the best implementations in terms of the least energy consumption in scalar and

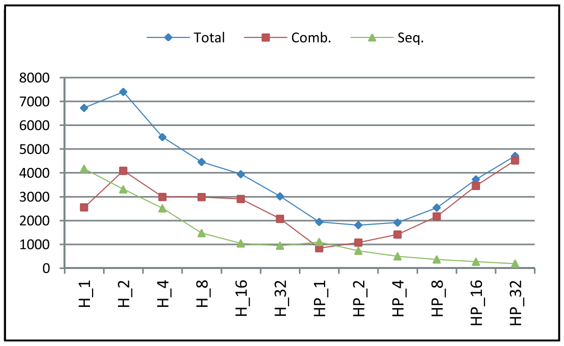


Figure 24. The energy trend (pJ) versus number of transformation rounds in each pipeline stage.

pipelined designs, the best pipelined design (*i.e.*, HP_2) consumes about 60% of the best scalar one (*i.e.*, H_{32}).

8.4. The optimum design

Determining the optimum design is dependent on the critical metric(s). Therefore, it is a question of what is the most critical performance metric (*i.e.*, speed, area, power, and energy) that should be considered. The answer begins with the intended application of the lightweight ciphers targeted for the low-resource devices. In this case, the design area and energy are the most critical factors for the hardware designs. And with the continuing miniaturization of transistor feature, the energy metric is becoming the more emphasized and dominant factor to consider. As such, to select the optimum design, we will compare the implementations using the metric in Equation (18). This metric exclusively rewards the designs with the minimum area and energy and gives more attention to the designs with the minimal energy.

$$\begin{gathered} metric = E^{\mu} \times LE \\ \text{where } \mu \text{ is the energy emphasis factor} \end{gathered} \tag{18}$$

Figure 25 plots the changing curves for the scalar and pipelined implementations using the proposed metric in Equation (18) with various values of μ = 1.0, 1.2, 1.4, 1.6, 1.8, and 2.0. Varying the value of μ illustrates which design consistently performs well in the optimization metric with different energy emphasis. Evidently,

- with higher emphasis on energy (μ>1.5), the optimum implementations are HP_2 and HP_4;
- with lower emphasis on energy (1<μ<1.5), the optimum implementation is H_1;
- among scalar implementations, the optimum designs are H_8 and H_1; and
- among pipeline implementations, the optimum designs are always HP_2 and HP_4.

Hence, as far as the low-resource designs are concerned, the pipeline designs have better performance compared

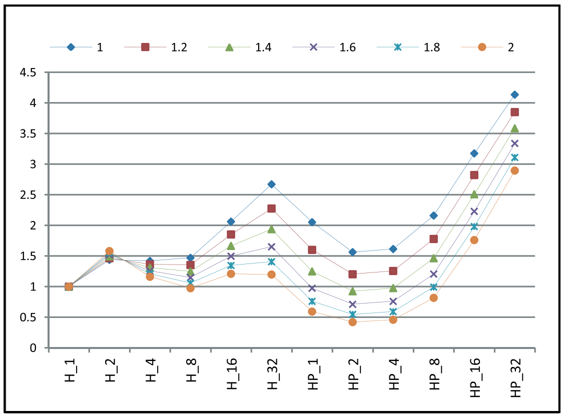


Figure 25. $E^{\mu} \times$ LE metric.

with the scalar ones. Overall, the best implementations are HP_2, HP_4, H_8, and H_1.

9. CONCLUSION

In this research work, we designed and optimized various scalar and pipelined FPGA implementations for the lightweight HIGHT cipher. Initially, the scalar design with one round was implemented and optimized. Then, multiple round scalar designs were implemented and analyzed. Next, the pipelined designs were implemented and studied. Various models to estimate number of LEs, power, and energy were developed to serve as guidelines for selecting the optimal design based on the underlying application domain of interest.

Considering the throughput metric, the best pipelined implementations were HP_1 and HP_2, whereas H_4 and H_8 were the best for the scalar ones. As for the best design in terms of the least number of LEs and power dissipation, H_1 was the best. In terms of the energy dissipation, the HP_2 pipelined design had the lowest energy consumption of all pipelined designs, whereas H_{32} for the scalar implementations. As for balancing the number of LEs and energy metrics, the pipeline implementations have superior results, and the best implementations were HP_2, HP_4, H_1, and H_8.

When comparing the best implementations of pipeline and scalar implementations, the scalar design requires 18% less resources and 10% less power, while the pipeline design had 18 times higher throughput and 60% less energy consumption.

Future research should develop generic performance model for lightweight ciphers. By providing algorithm parameters (*e.g.*, number of rounds, key generation, and round complexity), the performance model would predict resources, speed, power, and energy of FPGA design. Such a model would also assist researchers in developing low-energy secure lightweight ciphers.

Future research should also consider in-depth study of FPGA implementations of other lightweight ciphers such as Prince (2012) [21], LED (2011) [22], PRINTcipher [23], KATAN (2009) [13], PRESENT (2007) [7], and mCrypton (2005) [25]. Lightweight ciphers offer different levels of security and performance results. Eventually, mature reconfigurable security system could include a library of ciphers, where the FPGA hardware is reconfigured dynamically based on the desired security level and available resources.

REFERENCES

1. Atzori L, Iera A, Morabito G. The internet of things: a survey. *Computer networks* 2010; **54**(15): 2787–2805.
2. Rolfes C, Poschmann A, Leander G, Paar C. Ultra-lightweight implementations for smart devices–security for 1000 gate equivalents. In *Smart Card Research and Advanced Applications*. Springer, 2008; 89–103.
3. Zhang GH, Poon CCY, Zhang YT. A review on body area networks security for healthcare. *ISRN Communications and Networking* 2011; **2011**: 21.
4. Mohd BJ, Hayajneh T, Shakir MZ, Qaraqe KA, Vasilakos AV. Energy model for light-weight block ciphers for WBAN applications. In *2014 EAI 4th International Conference on Wireless Mobile Communication and Healthcare (Mobihealth),* 2014 Nov 3, IEEE, 2014; 1–4.
5. Kerckhof S, Durvaux F, Hocquet C, Bol D, Standaert F-X. Towards green cryptography: a comparison of lightweight ciphers from the energy viewpoint. In *Cryptographic Hardware and Embedded Systems–CHES 2012*. Springer, 2012; 390–407.
6. Piedra A, Braeken A, Touhafi A. Extending the IEEE 802.15. 4 ecurity Suite with a Compact Implementation of the NIST P-192/B-163 Elliptic Curves. *Sensors* 2013; **13**(8): 9704–9728.
7. Bogdanov A, Knudsen LR, Leander G, *et al.* *PRESENT: An Ultra-lightweight Block Cipher*. Springer: Berlin Heidelberg, 2007.
8. Cazorla M, Marquet K, Minier M. Survey and benchmark of lightweight block ciphers for wireless sensor networks. *IDEA* 2013; **64**(128): 34.
9. Wollinger T, Guajardo J, Paar C. Security on FPGAs: state-of-the-art implementations and attacks. *ACM Transactions on Embedded Computing Systems (TECS)* 2004; **3**(3): 534–574.
10. Bossuet L, Grand M, Gaspar L, Fischer V, Gogniat G. Architectures of flexible symmetric key crypto engines—a survey: from hardware coprocessor to multi-crypto-processor system on chip. *ACM Computing Surveys (CSUR)* 2013; **45**(4): 41.
11. Hong D, Sung J, Hong S, Lim J, Lee S, Koo BS, Lee C, Chang D, Lee J, Jeong K, Kim H. HIGHT: a new block cipher suitable for low-resource device. In *Cryptographic Hardware and Embedded Systems-CHES 2006*. Springer, 2006; 46–59.
12. Knudsen LR. Practically secure feistel ciphers. In *Fast software encryption*. Springer, 1994; 211–221.
13. De Canniere C, Dunkelman O, Knežević M. KATAN and KTANTAN–a family of small and efficient hardware-oriented block ciphers. In *Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer, 2009; 272–288.
14. Eisenbarth T, Kumar S, Paar C, Poschmann A, Uhsadel L. A survey of lightweight-cryptography implementations. *IEEE Design & Test of Computers* 2007; **24**(6): 522–533.
15. Eisenbarth T, Gong Z, Güneysu T, *et al.* Compact implementation and performance evaluation of block ciphers in ATtiny devices. In *Progress in cryptology-africacrypt 2012*. Springer, 2012; 172–187.
16. Lim Y-Il, Lee J-H, You Y, Cho K-R. Implementation of HIGHT cryptic circuit for RFID tag. *IEICE Electronics Express* 2009; **6**(4): 180–186.
17. Yalla P, Kaps J. Lightweight Cryptography for FPGAs. *International Conference on Reconfigurable Computing and FPGAs, 2009*, Cancun, Mexico, 2009; 225–230.
18. Kolay S, Mukhopadhyay D. Khudra: a new lightweight block cipher for FPGAs. In *Security, Privacy, and Applied Cryptography Engineering*, vol. 8804, Chakraborty R, Matyas V, Schaumont P (eds), Lecture Notes in Computer Science. Springer International Publishing, 2014; 126–145.
19. Lee D, Kim D-C, Kwon D, Kim H. Efficient hardware implementation of the lightweight block encryption algorithm LEA. *Sensors* 2014; **14**(1): 975–994.
20. Engels D, Saarinen M-JO, Schweitzer P, Smith EM. The Hummingbird-2 lightweight authenticated encryption algorithm. In *Rfid. Security and Privacy*. Springer: Amherst, Massachusetts (USA), 2012; 19–31.
21. Borghoff J, Canteaut A, Güneysu T, *et al.* PRINCE–a low-latency block cipher for pervasive computing applications. In *Advances in cryptology–asiacrypt 2012*. Springer, 2012; 208–225.
22. Guo J, Peyrin T, Poschmann A, Robshaw M. The LED block cipher. In *Cryptographic Hardware and Embedded Systems–CHES 2011*. Springer, 2011; 326–341.
23. Knudsen L, Leander G, Poschmann A, Robshaw MJB. PRINTcipher: a block cipher for IC-printing. In *Cryptographic Hardware and Embedded Systems, CHES 2010*. Springer, 2010; 16–32.

24. Engels D, Fan X, Gong G, Hu H, Smith EM. Ultra-lightweight cryptography for low-cost RFID tags: hummingbird algorithm and protocol. *Centre for Applied Cryptographic Research (CACR) Technical Reports* 2009; **29**.
25. Lim CH, Korkishko T. mCrypton—a lightweight block cipher for security of low-cost RFID tags and sensors. In *Information security applications*. Springer, 2006; 243–258.
26. Wheeler DJ, Needham RM. TEA, a tiny encryption algorithm, 1995; 363–366.
27. Needham RM, Wheeler DJ. Tea extensions. *Report, Cambridge University*, Cambridge, UK (October 1997), 1997.
28. Lee J-H, Lim DG. Parallel architecture for high-speed block cipher, HIGHT. *International Journal of Security and Its Applications* 2014; **8**(2): 59–66.
29. Wu W, Zhang L. LBlock: a lightweight block cipher. In *Applied cryptography and network security*. Springer, 2011; 327–344.
30. Badel S, Dağtekin N, Nakahara Jr J, *et al.* ARMADILLO: a multi-purpose cryptographic primitive dedicated to hardware. In *Cryptographic Hardware and Embedded Systems, CHES 2010*. Springer, 2010; 398–412.
31. Rahimunnisa K, Karthigaikumar P, Rasheed S, Jayakumar J, SureshKumar S. FPGA implementation of AES algorithm for high throughput using folded parallel architecture. *Security and Communication Networks* 2014; **7**(11): 2225–2236.
32. Kundi D-S, Aziz A, Kazmi M. An efficient single unit T-box/T- 1-box implementation for 128-bit AES on FPGA. *Security and Communication Networks* 2015; **8**(9): 1725–1731.
33. Aoki K, Ichikawa T, Kanda M, *et al.* Camellia: a 128-bit block cipher suitable for multiple platforms. *NESSIE Algorithm Submission* 2000.
34. Shibutani K, Isobe T, Hiwatari H, Mitsuda A, Akishita T, Shirai T. Piccolo: an ultra-lightweight blockcipher. In *Cryptographic Hardware and Embedded Systems–CHES 2011*. Springer, 2011; 342–357.
35. Kolay S, Mukhopadhyay D. Khudra: a new lightweight block cipher for fpgas. In *Security, Privacy, and Applied Cryptography Engineering*. Springer, 2014; 126–145.
36. Zhang X, Heys HM, Li C. Fpga implementation and energy cost analysis of two light-weight involutional block ciphers targeted to wireless sensor networks. *Mobile Networks and Applications* 2013; **18**(2): 222–234.
37. Mohd BJ, Hayajneh T, Abed S, Itradat A. Analysis and modeling of fpga implementations of spatial steganography methods. *Journal of Circuits, Systems, and Computers* 2014; **23**(02): 1450018.
38. Mohd BJ, Hayajneh T, Quttoum AN. Wavelet-transform steganography: algorithm and hardware implementation. *International Journal of Electronic Security and Digital Forensics* 2013; **5**(3-4): 241–256.
39. ftp://ftp.altera.com/up/pub/Altera_Material/11.0/Tutorials/Verilog/Quartus_II_introduction.pdf.
40. Abed S, Mohd BJ, Al-bayati Z, Alouneh S. Low power wallace multiplier design based on wide counters. *International Journal of Circuit Theory and Applications* 2012; **40**(11): 1175–1185.
41. Mohd BJ, Abed S, Al-Hayajneh T, Alouneh S. FPGA hardware of the LSB steganography method. *International Conference on Computer, Information and Telecommunication Systems (CITS), 2012*, IEEE, 2012; 1–4.
42. Mohd BJ, Hayajneh T, Khalaf ZA. Optimization and modeling of FPGA implementation of the Katan Cipher. *International Conference on Information and Communication Systems (ICICS), 2015 6th*, IEEE, 2015; 68–72.
43. Mohd BJ, Hayajneh T, Abu Khalaf Z, Vasilakos AV. A comparative study of steganography designs based on multiple FPGA platforms. *International Journal of Electronic Security and Digital Forensics*, inpress.
44. Neil HE, Harris D. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publisher, 2005.