Efficient FPGA Implementations of PRINT$_{CIPHER}$

Tadashi Okabe
Information Technology Group
Tokyo Metropolitan Industrial Technology Research Institute, Tokyo, Japan

Abstract — This article presents field programmable gate array (FPGA) implementations of PRINT$_{CIPHER}$, which is an encryption algorithm for protecting important data in restricted computing environments such as lightweight devices and Radio Frequency Identifier (RFID) cards. PRINT$_{CIPHER}$ has been designed to ensure integrity and authentication with minimum resources. This article is the first work about FPGA implementations of PRINT$_{CIPHER}$ and proposes firstly efficient hardware architectures for PRINT$_{CIPHER}$. Hardware implementation results for these architectures have been evaluated in this article. The proposed architectures are synthesizable and implemented on Spartan-3, Spartan-6 and Artix-7 Xilinx FPGAs.

Keywords — Lightweight block cipher, Hardware implementation, FPGA, Hardware architecture, PRINT$_{CIPHER}$

I. INTRODUCTION

Currently, the Internet of Things (IoT) paradigm connects all industrial and consumer electronics manufacturers to worldwide communication networks. In the IoT society, both general purpose computers and lightweight portable electronics need higher security. Regarding security, standard block ciphers such as Triple DES [1] or AES [2] are readily available. However, these ciphers cannot be applied to small, lightweight devices that are resource-constrained or have integrated circuits (ICs) with low processing power. However, many lightweight block ciphers are now being proposed for use with radio frequency identification (RFID) tags, small field programmable gate arrays (FPGAs), micro-controllers, and other lightweight devices.

PRINT$_{CIPHER}$ is a lightweight block cipher designed for IC printing and efficient hardware implementations in resource-constrained compact devices. It is a substitution-permutation network (SPN) structure with a substitution layer that consists of sixteen 3-bit SBOXes. Its permutation layer is the linear diffusion of simple bit permutation. PRINT$_{CIPHER}$ has 48- or 96-bit block sizes for plaintext or ciphertext, and 80- or 160-bit key sizes for the cipher key, and the number of rounds is 48 or 96. The cipher key is divided into two parts: the first is the XOR function with intermediate round data and the second is the key permutation function. All components of PRINT$_{CIPHER}$ easily fit into the 4-bit or 6-bit input lookup table (LUT; LUT is 4-bit or 6-bit input function generators and constitute the basic building blocks of most recent reconfigurable devices) of FPGAs, and its key scheduling allows identical round keys in each round. In addition to its compact efficiency in hardware implementation, the identical round key at each round and the small 3-bit SBOX speeds up the PRINT$_{CIPHER}$ encryption process. In practice, PRINT$_{CIPHER}$ has potential applications for resource-constrained compact crypt devices and high throughput data encryption fields.

This article presents FPGA implementations of PRINT$_{CIPHER}$ and compares the performances with recent lightweight block ciphers such as PRESENT, SIMON and several others. The first presented architecture computes one round per clock cycle, while the second is based on the 3-bit word-serial architecture presented in PRINT$_{CIPHER}$3 for application-specific integrated circuits (ASICs). Our main contribution is the architectures that are also 3-bit word-serial by nature and perform PRINT$_{CIPHER}$ computations based on finite state machine (FSM) models. The first architecture (one round per clock cycle) is actually better than the second one (3-bit word-serial), because it achieves lower area, lower power and better throughput. For this purpose, we investigated various contexts (round iterative rolled and word-serialized iterative rolled implementations, with or without a register key) on the recent Xilinx FPGAs on Spartan-3, Spartan-6 and Artix-7 platforms.

This article is organized as follows. In Section 2, we briefly present the specifications of the PRINT$_{CIPHER}$ encryption algorithm. Section 3 describes the FPGA design methodology. In Section 4 we describe the implementation results of the proposed hardware architectures and present comparisons with other lightweight block ciphers. Finally, we discuss the conclusions of the study and future work in Section 5.

II. PRINT$_{CIPHER}$ SPECIFICATIONS

In this section, we describe the specifications and data encryption algorithm of PRINT$_{CIPHER}$.

A Parameters

PRINT$_{CIPHER}$ operates on 48- or 96-bit plaintext, ciphertext or intermediate data blocks and 80- or 160-bit cipher key blocks. It is an iterative block cipher based on the repetition of 48 or 96 identical key-dependent round functions. More detailed descriptions are given in the following subsections.

B Round Function

The round function is described in Fig. 1, where we distinguish linear diffusion layers and a nonlinear layer. The linear diffusion layer is built from bitwise key additions (denoted as the KEYXOR Layer in Fig. 1), bit permutations (48-bit wire crossings, denoted as pLayer), sub-key dependent permutations layer (sub-key selected 3-bit wire crossings, denoted as the...
KEY_PERM Layer), and least significant 6/7-bitwise round constant additions (denoted as the RC_XOR Layer). The nonlinear layer is built from the parallel application of 3x3 substitution boxes to the intermediate state. For FPGA resource efficiency purposes, these substitution boxes are the same and are constructed from the smaller 3-bit SBOX.

Each round consists of the following five functions:

1. Key XOR with each round input data
2. Linear diffusion
3. Right-most 6/7-bit intermediate data, XORed with round counter constant
4. Key-dependent permutation
5. Nonlinear transformation with SBOX

Each 3-bit word connection in the round function is depicted in Fig. 2. Subsections B.1 to B.5 describe each function layer in detail.

![Figure 1 Round function of encryption process Flow](image)

![Figure 2 Round Function of PRINTCIPHER-48](image)
B.1. **Key_XOR Layer** This function layer is simply bitwise XOR (exclusive-or) between the current state and 48/96-bit sub-key, sk1. This sub-key is identical in all rounds and is the most significant 48/96-bit sub-key of the cipher key.

B.2. **Linear diffusion layer** Linear diffusion is described in Fig. 3 as a simple bit permutation layer. Bit $i$ of the current state is controlled by the following equation

$$P(i) = \begin{cases} 
3 \times i \mod b - 1 & \text{for } 0 \leq i \leq b - 2 \\
b - 1 & \text{for } i = b - 1
\end{cases}$$  \hspace{1cm} (1)

Where $b$ is the bit size of the cipher blocks, $b \in \{48, 96\}$. In this article, we use the word “pLayer” as the function $P(i)$.

B.3. **Round counter** The round counter, RC, is specified in the following equation. This counter is the linear feedback shift register (LFSR) sequence and its initial value is 1(hex).

$$t = 1 + x_{n-1} + x_{n-2}$$

$$(2)$$

where $x_i$ is the current state at round $i$, and $n$ is the number of rounds.

B.4. **RC_XOR Layer** This function layer is simply a bitwise XOR (exclusive-or) between the current state and the round counter in equation 2. In this layer, we XORed between the least significant 6/7 bits of intermediate data and the round counter state.

B.5. **KEY_PERM Layer** Keyed permutation is illustrated in Fig. 4, which is a selector logic block in which the 2-bit sub-key selects the 3-bit input to the 3-bit output corresponding to the permutation law. The permutation law is depicted in Fig. 4. For the keyed permutation layer, 48/96-bit permutations (of each three bits) are picked from a set of four in a key-dependent manner.
**B.6. SBOX Layer** The SBOX is described in Table 1. Although most other lightweight block ciphers use a input 4-bit to output 4-bit SBOX, the PRINTCIPHER SBOX is particularly input 3-bit to output 3-bit. This is reasonable for a compact FPGA implementation to fit the nonlinear transformation to LUTs on FPGAs in technology mapping. For the SBOX Layer, the current state is 16/32 3-bit words. For each word an identical SBOX is used.

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>S(x)</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

**C Key Schedule (Round Key and Permutation Key Generation)**

The key scheduling process is not used in PRINTCIPHER. In each round, the most significant 48/96 bits of the cipher key are the round key that does exclusive-or with round data in the KEY_XOR function. On the other hand, the least significant 32/64 bits of the cipher key is the permutation key, which is a selection flag signal in the KEY_PERM function. Thus, round key and permutation key generation is very simple in the PRINTCIPHER encryption algorithm.

Accordingly, PRINTCIPHER allows encryption/decryption with exactly the same round keys, sk1 and sk2. This means that the hardware of the key schedule is very simple (only the dividing round key and permutation key) and the storage of round keys is not necessary.

**D Encryption (Decryption) Process**

The complete cipher consists of the iterative 48/96 rounds discussed in subsections B.1 to B.5. Due to the identical round key structures of PRINTCIPHER, the key scheduling function only executes in the first encryption (decryption) process. In pseudo C code, we have:

```c
Input : plaintext, key;
Output : ciphertext;

PRINTcipher48-encrypt(plaintext, key)
{
    state = plaintext;
    Key_Schedule(key);
    for (i=0; i<48; i++) Round(state, sk1, sk2, i);
    ciphertext = state;
}

Key_Schedule(key)
{
    sk1 = key[80:32];
    sk2 = key[31:0];
}

Round(state, sk1, sk2, i)
{
    KEY_XOR_Layer(state, sk1);
    pLayer(state);
    RC_XOR_Layer(state, i);
    KEY_PERM_Layer(state, sk2);
    SBOX_Layer(state);
}
```

where key is the 80/160-bit input cipher key, state is the 48/96-bit intermediate round data block and cipher text data after the last round, sk1 is the 48/96-bit round key when it is used in KEY_XOR layer, sk2 is the 32/64-bit permutation key when it is used in the KEY_PERM Layer. The PRINTCIPHER decryption process is simply the reverse of encryption. More details about this particular encryption/decryption algorithm are available in the PRINTCIPHER original paper [3].

**III. DESIGN METHODOLOGY**

FPGAs are usually composed of reconfigurable logic blocks combined with LUT block memories and arithmetic circuits [4], [5], [6]. The basic logic blocks of FPGAs include a 4/6-input LUT and a storage element, such as an flip flop (FF) or register. Most FPGA manufacturers provide users with fast carry logic and high-performance digital signal processor (DSP) logic blocks, shift registers, and so on.
As reconfigurable components are divided into logic elements and storage elements, efficient implementation is a compromise between both combinatorial logic and sequential logic, and resulting performances and power consumptions. These viewpoints lead to different definitions of implementation efficiency. In this article, we selected the following efficiencies: the first two are used in [7], and the third efficiency, called new power regulation efficiency, is our proposal.

1) In terms of area performances, the efficiency of a block cipher is given by the ratio of throughput (Mbps) to area (slices). The larger the ratio, the better the performance efficiency per unit area.
2) In terms of resource performance, the efficiency is easily tested by computing the ratio of the number of registers to the number of LUTs. Ideally, the ratio should be close to one the number of LUTs. Ideally, the ratio should be close to one. When this value is a value close to one, the circuit resource is more efficient.
3) In terms of power regulation performance, the power efficiency of a block cipher is the ratio of power regulation (mW) to area (slices). When this value is smaller, the power consumption per unit area is smaller.

PRINTCipher is designed to allow IC printing and very efficient FPGA implementations. Our proposed architectures are iterative round function architecture or word-serial architecture, divided into element sub-functions from the round function. These are defined to maximize the hardware efficiency or throughput, and minimize the power regulation, with trade-offs among these performance indexes.

Finally, depending on the optimization criteria, different architectures can be employed. Although optimization for maximum processing speed can ideally be achieved by a fully pipeline architecture, we did not select this architecture for applying PRINTCipher to lightweight devices. In the applications requiring minimum area or power regulation, a loop architecture with only one round or division into element functions in round implementations seems to be the best choice. In this study, we tried to maximize the previously defined efficiencies. In the following sections, we present the results of various FPGA implementations of PRINTCipher.

I. IMPLEMENTATION

In this section, we investigate the practical implementation of different architectures for PRINTCipher. All the architectures proposed in this section allow the choice of default parameters for every implementation. The area and frequency estimations presented are provided after implementation with Xilinx ISE 14.7 on the Xilinx Spartan-3, Spartan-6 and Artix-7 platforms. The timing constraints were applied according to the reference clock.

A. Round Iterative Roll Architectures

For standard lightweight block cipher applications, we propose the round iterative roll architecture with round-based implementation (see Fig. 5). We applied two key scheduling strategies. If a minimum area is required, a hard-wired key scheduling implementation is provided. On the other hand, if maximum throughput is required, the encryption key input is stored in the registers. We studied two key scheduling models, Std. I and Std. II, for the round iterative roll architecture (see Fig. 6), and selected a simple finite state machine description with Verilog-HDL.

Figure 5 Round iterative roll architecture
B. Word Serial Roll Architecture

In the applications requiring minimum area and minimum power regulation, we propose the 3-bit word serial roll architecture (see Fig. 7). To decrease the occupied area and power consumption requirements, we only considered the 3-bit iterative FSM strategy structured by sub-functions in a round. In addition to the efficiency advantages already mentioned, 3-bit word serial structures are especially convenient for loop architectures because they allow the combination of the keyed permutation function with the SBox function in the round. Our proposed FSM is illustrated in Figs. 8 to Fig. 10. For word serial rolled architectures, it is possible to use the FPGA LUT blocks to implement the round SBOX, XOR logic, diffusion layer and round counter. We selected the 3-bit word bus because the keyed permutation and SBOX have a 3-bit input/output signal in PRINTCIPHER. We also selected the 48-bit bus width in pLayer to optimize the trade-offs between the processing cycle and logic resources usage on FPGAs. We proposed five types of FSM models. We divided them into several sub-states corresponding to the functions in a round, and we made the sub-states as small as possible in occupied area and power consumption. The proposed-I model describes five simple states divided into each function, and proposed-II merges KEY_XOR, pLayer and RC_XOR functions of the proposed-I model into one “KEY_XOR function. The proposed-III model combines KEY_PERM and SBOX. The proposed-IV model merges pLayer and RC_XOR functions from proposed-III. Finally, the proposed-V model is combined KEY_XOR, pLayer and RC_XOR.

Figure 7 Word serial roll architecture

Figure 6.1 cycle Round iterative roll FSM diagram (Std. I and Std. II)

Figure 8 16 cycles word serial roll FSM diagram (Std. III and Std. IV)
II. RESULT AND COMPARISON WITH OTHER LIGHTWEIGHT BLOCK CIPHERS

Our implementation results are presented in Table 2, where the FPGA resource requirements (in LUTs and slices) corresponding to an area, the maximum operation frequency, the maximum throughput and the power regulation are provided. All experimental results were extracted after place and route with ISE Design Suite version 14.7 from Xilinx Inc. on Spartan-3 XC3S700A, Spartan-3A XC3S500A and Spartan-3 XC3S200A. For the Spartan-3 XC3S200A, the area was significantly lower compared to Spartan-3A XC3S500A for the proposed cipher.

Figure 9 51/48/35 cycles Word serial roll FSM diagram (Proposed-I/II/III)

Figure 10 33/32 cycles Word serial roll FSM diagram (Proposed-IV/V)
6 XC6SLX45, and Artix-7 XC7A100 platforms with speed grades -5/-3/3/-3, respectively. From Table 2, we note that the area-optimized PRINTCIPHER encryption core (Proposed-V) can achieve a throughput of 3.83 Mbps at the cost of 91 slices on Spartan-3 FPGA platform, whereas the round-rolled iterative encryption core (Std. II) occupies 139 slices and operates at 8.46 Mbps on the same platform.

Table 3 describes the performance comparison of our PRINTCIPHER implementation with existing FPGA implementations of block ciphers LED [13], PRESENT [8], [9], [10], HIGHT [10], AES [11], [12], ICEBERG [7], SEA [14], XTEA [15], SIMON [16] and CLEFFIA [17]. Note that numerous PRESENT, SIMON, and AES hardware architectures have been proposed in the literature, but we only use those implementations on low-cost Spartan and Artix series FPGA devices with speed grades -5 or -3 and above for the purpose of comparison. Moreover, the implementation figures of ICEBERG, SEA and CLEFFIA are only available on Virtex-II or Virtex-4 series FPGAs. We also would like to point out that it is quite difficult to provide a fair comparison among different implementations on FPGAs by taking into account the diversity of FPGA devices and packages, speed grade level, and synthesis and implementation tools. Therefore, we include additional information such as the implementation platform and speed grade level in Table 3.

Our experimental results show that in the low-cost FPGA implementations, PRINTCIPHER can achieve larger throughput with a smaller area requirement, when compared to block ciphers PRESENT, XTEA, SEA and AES. However, the implementation of the ultra-lightweight block cipher SIMON is more efficient than that of PRINTCIPHER. The main reason is the complex internal encryption process in the PRINTCIPHER encryption algorithm. As a result, the encryption unit is more complicated and the delay of the critical path is much longer in the PRINTCIPHER hardware architecture. Although the area of PRINTCIPHER is as compact as that of HIGHT on the Spartan-3 FPGA platform, the throughput of PRINTCIPHER is lower than that of HIGHT. The reason is that PRINTCIPHER has many rounds in the encryption process, and this leads to slower frequency and less throughput than those of HIGHT. Additionally, though an occupied area (slices) of PRINTCIPHER is as compact as the LED area on the Artix-7 FPGA platform, PRINTCIPHER throughput is less than LED throughput, but PRINTCIPHER has a sufficiently small area, lower power regulation and higher throughput on lightweight cipher applications.

Table 2 FPGA implementation results of PRINTCIPHER block cipher with different architectures

<table>
<thead>
<tr>
<th>Architectures</th>
<th>Devices</th>
<th>Area (Resources)</th>
<th>Speed</th>
<th>Power</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Registers</td>
<td>LUTs</td>
<td>Slices</td>
<td>Latency (cycle)</td>
</tr>
<tr>
<td>Std. I</td>
<td></td>
<td>139</td>
<td>262</td>
<td>180</td>
<td>768</td>
</tr>
<tr>
<td>Std. II</td>
<td></td>
<td>59</td>
<td>261</td>
<td>139</td>
<td>768</td>
</tr>
<tr>
<td>Std. III</td>
<td></td>
<td>135</td>
<td>332</td>
<td>224</td>
<td>48</td>
</tr>
<tr>
<td>Std. IV</td>
<td></td>
<td>55</td>
<td>396</td>
<td>210</td>
<td>48</td>
</tr>
<tr>
<td>Proposed-I</td>
<td>Spartan-3</td>
<td>68</td>
<td>233</td>
<td>121</td>
<td>2448</td>
</tr>
<tr>
<td>Proposed-II</td>
<td>Spartan-4</td>
<td>65</td>
<td>238</td>
<td>128</td>
<td>2304</td>
</tr>
<tr>
<td>Proposed-III</td>
<td>Spartan-6</td>
<td>63</td>
<td>222</td>
<td>113</td>
<td>1584</td>
</tr>
<tr>
<td>Proposed-IV</td>
<td>Spartan-7</td>
<td>69</td>
<td>233</td>
<td>123</td>
<td>1680</td>
</tr>
<tr>
<td>Proposed-V</td>
<td>Artix-7</td>
<td>61</td>
<td>174</td>
<td>91</td>
<td>1536</td>
</tr>
<tr>
<td>Std. I</td>
<td></td>
<td>143</td>
<td>182</td>
<td>90</td>
<td>768</td>
</tr>
<tr>
<td>Std. II</td>
<td></td>
<td>62</td>
<td>154</td>
<td>48</td>
<td>768</td>
</tr>
<tr>
<td>Std. III</td>
<td></td>
<td>135</td>
<td>270</td>
<td>111</td>
<td>48</td>
</tr>
<tr>
<td>Std. IV</td>
<td></td>
<td>55</td>
<td>270</td>
<td>114</td>
<td>48</td>
</tr>
<tr>
<td>Proposed-I</td>
<td>Spartan-6</td>
<td>68</td>
<td>125</td>
<td>42</td>
<td>2448</td>
</tr>
<tr>
<td>Proposed-II</td>
<td>Spartan-7</td>
<td>66</td>
<td>107</td>
<td>34</td>
<td>2304</td>
</tr>
<tr>
<td>Proposed-III</td>
<td>Spartan-8</td>
<td>69</td>
<td>99</td>
<td>35</td>
<td>1584</td>
</tr>
<tr>
<td>Proposed-IV</td>
<td>Spartan-9</td>
<td>66</td>
<td>135</td>
<td>41</td>
<td>1680</td>
</tr>
<tr>
<td>Proposed-V</td>
<td>Artix-7</td>
<td>65</td>
<td>116</td>
<td>43</td>
<td>1536</td>
</tr>
<tr>
<td>Std. I</td>
<td></td>
<td>140</td>
<td>163</td>
<td>77</td>
<td>768</td>
</tr>
<tr>
<td>Std. II</td>
<td></td>
<td>60</td>
<td>153</td>
<td>64</td>
<td>768</td>
</tr>
<tr>
<td>Std. III</td>
<td></td>
<td>135</td>
<td>280</td>
<td>151</td>
<td>48</td>
</tr>
<tr>
<td>Std. IV</td>
<td></td>
<td>55</td>
<td>276</td>
<td>139</td>
<td>48</td>
</tr>
<tr>
<td>Proposed-I</td>
<td></td>
<td>61</td>
<td>117</td>
<td>40</td>
<td>2448</td>
</tr>
<tr>
<td>Proposed-II</td>
<td></td>
<td>61</td>
<td>100</td>
<td>33</td>
<td>2304</td>
</tr>
<tr>
<td>Proposed-III</td>
<td></td>
<td>61</td>
<td>95</td>
<td>29</td>
<td>1584</td>
</tr>
<tr>
<td>Proposed-IV</td>
<td></td>
<td>61</td>
<td>114</td>
<td>38</td>
<td>1680</td>
</tr>
<tr>
<td>Proposed-V</td>
<td></td>
<td>60</td>
<td>112</td>
<td>38</td>
<td>1536</td>
</tr>
</tbody>
</table>
I. CONCLUSION

In this article, we presented the first efficient FPGA implementations of PRINTCIPHER and analyzed the feasibility of compact and low power implementations of PRINTCIPHER on FPGAs. We proposed some round-based and word-serial architectures and evaluated their efficiencies. The proposed area-optimized PRINTCIPHER encryption core can encrypt a 48-bit plaintext block with 2.67 Mbps throughput (2,304 clock cycles) on Spartan-6. Moreover, we implemented several possible trade-offs when executing encryption round processes. In particular, we confirmed the efficiency of a proposed word-serialization model in low-cost and low-power aspects. Compared to other lightweight block cipher implementations on FPGAs, XTEA, ICEBERG, SEA, SIMON, LED and AES, PRINTCIPHER can achieve larger throughput with smaller area requirements. Our results show that PRINTCIPHER is a good encryption algorithm for resource-constrained devices. Consequently, PRINTCIPHER can be considered to be an ideal cryptographic primitive for resource-constrained environments. In future work, we will investigate the resistance against side-channel attacks in our implementations and develop countermeasures to maintain resistance and reliability against these attacks.

Table 3 FPGA implementation results of other block ciphers

<table>
<thead>
<tr>
<th>Block Ciphers</th>
<th>Devices</th>
<th>Area (Resources)</th>
<th>Speed</th>
<th>Power (mW)</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LUTs</td>
<td>Slices</td>
<td>Latency [cycles]</td>
<td>Max Freq [MHz]</td>
</tr>
<tr>
<td>PRESENT-1 [8]</td>
<td>Spartan-3 XC3S500</td>
<td>—</td>
<td>271</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>PRESENT [9]</td>
<td>Spartan-3 XC3S400-5</td>
<td>—</td>
<td>202</td>
<td>32</td>
<td>254</td>
</tr>
<tr>
<td>PRESENT [10]</td>
<td>Spartan-3 XC3S50-5</td>
<td>—</td>
<td>117</td>
<td>256</td>
<td>114.8</td>
</tr>
<tr>
<td>AES [12]</td>
<td>Spartan-3 XC3S50-5</td>
<td>—</td>
<td>184</td>
<td>160</td>
<td>45.6</td>
</tr>
<tr>
<td>HIGHT [10]</td>
<td>Spartan-3 XC3S50-5</td>
<td>—</td>
<td>91</td>
<td>—</td>
<td>163.7</td>
</tr>
<tr>
<td>Artix-7 XC7A100T-3</td>
<td>—</td>
<td>78</td>
<td>37</td>
<td>1120</td>
<td>378</td>
</tr>
<tr>
<td>XTEA [15]</td>
<td>Spartan-3 XC3S50-5</td>
<td>—</td>
<td>254</td>
<td>112</td>
<td>62.6</td>
</tr>
<tr>
<td>SIMON [16]</td>
<td>Spartan-3 XC3S500</td>
<td>72</td>
<td>36</td>
<td>—</td>
<td>136</td>
</tr>
<tr>
<td>CLEFIA [17]</td>
<td>Virtex-4</td>
<td>238</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

REFERENCES


