Available online at www.sciencedirect.com
ScienceDirect
INTEGRATION, the VLSI journal 40 (2007) 20–27
INTEGRATION the VLSI journal
www.elsevier.com/locate/vlsi

FPGA implementations of the ICEBERG block cipher

F.-X. Standaert*, G. Piret, G. Rouvroy, J.-J. Quisquater

UCL Crypto Group, Place du Levant, 3, B-1348 Louvain-La-Neuve, Belgium

Abstract

This paper presents field programmable gate array (FPGA) implementations of ICEBERG, a block cipher designed for reconfigurable hardware implementations and presented at FSE 2004. All its components are involutional and allow very efficient combinations of encryption/decryption. The implementations proposed also allow changing the key and encrypt/decrypt (E/D) mode for every plain text, without any performance loss. In comparison with other recent block ciphers, the implementation results of ICEBERG show a significant improvement of hardware efficiency. Moreover, the key and E/D agility allow considering new encryption modes to counteract certain side-channel attacks.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Block ciphers; Hardware implementations; FPGAs

1. Introduction

In October 2000, National Institute of Standards and Technology (NIST) selected Rijndael as the new Advanced Encryption Standard. The selection process included performance evaluation on both software and hardware platforms. However, as implementation versatility was a criteria for the selection of the AES, it appeared that Rijndael was not optimal for reconfigurable hardware implementations. For example, its highly expensive substitution boxes may be a bottleneck in certain contexts. The combination of encryption and decryption in hardware is usually another critical point (e.g. see [14–16,21]).

ICEBERG is a block cipher designed for efficient reconfigurable hardware implementations. It is based on an involutional structure so that the forward and inverse operation of the cipher may be performed with exactly the same hardware. All its components easily fit into the 4-bit input lookup tables of FPGAs, and its key scheduling allows the round keys to be derived "on the fly" in encryption and decryption mode. In addition to hardware efficiency, the key and E/D agility allow considering new encryption modes to counteract certain side-channel attacks. In practice, very low-cost hardware crypto-processors and high-throughput data encryption are potential applications of ICEBERG.

This paper presents FPGA implementations of ICEBERG and compares their performances with the ones of recent block ciphers (e.g. AES and NESSIE candidates). Although ICEBERG implementations offer features that most block ciphers do not provide (e.g. key and E/D agility), its implementation results exhibit a significant improvement of hardware efficiency. For this purpose, we investigated various contexts (loop and unrolled implementations, with or without feedback) on the recent Xilinx Virtex-II® FPGAs.

The paper is structured as follows. Section 2 briefly presents the specifications of ICEBERG and Section 3 describes our FPGA design methodology. Section 4 lists the combinatorial cost of the block cipher components. The implementation results for various architectures are in Section 5 and comparisons with other block ciphers are in Section 6. Resistance against side-channel analysis is briefly discussed in Section 7. Finally, conclusions are in Section 8.

2. Specifications

2.1. Block and key size

ICEBERG operates on 64-bit blocks and uses a 128-bit key. It is an involutional iterative block cipher based on the

*Corresponding author.
E-mail addresses: standaert@dice.ucl.ac.be (F.-X. Standaert), piret@dice.ucl.ac.be (G. Piret), rouvroy@dice.ucl.ac.be (G. Rouvroy), quisquater@dice.ucl.ac.be (J.-J. Quisquater).

0167-9260/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.vlsi.2005.12.008

repetition of 16 identical key-dependent round functions. In the next subsections, we briefly present the algorithm. A more detailed description can be found in the original paper [1].

2.2. *The round function*

The round function is pictured in Fig. 1, where we distinguish a non-linear layer and a linear diffusion layer.

The *non-linear layer* is built from the parallel application of 8×8 substitution boxes to the cipher state. For efficiency purposes, these boxes are constructed from smaller 4×4 S-boxes $S0$, $S1$ and bit permutations $P8$ (i.e. 8-bit wire crossings).

The *linear diffusion layer* is built from bit permutations $P64$ (i.e. 64-bit wire crossings), bit permutations $P4$ (i.e. 4-bit wire crossings), bitwise key additions (denoted as $\oplus$ in the figure) and small 4×4 diffusion boxes D. These boxes perform a simple multiplication:

$$\begin{pmatrix} y_3 \\ y_2 \\ y_1 \\ y_0 \end{pmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \times \begin{pmatrix} x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix},$$

where every output bit is a $\oplus$ operation between three input bits. It is therefore efficiently combined with the key addition inside a single 4-input LUT.

2.3. *The key schedule*

The key scheduling process consists of key expansion and key selection.

The *key expansion* expands the cipher key K into a sequence of keys $K^0, K^1, \ldots, K^{16}$. We set the initial key $K^0 = K$. The following keys are obtained by a keyround function so that: $K^{i+1} = keyround(K^i)$.

The *keyround* is pictured in Fig. 2, where we distinguish a conditional shift layer, bit permutations $P128$ (i.e. 128-bit wire crossings) and S-boxes $S0$. The conditional shift operation depends on a round constant C that will be discussed further.

Finally, the *key selection* first performs a simple compression function that selects 64 bytes of K^i having odd indices. Thereafter, a 4×4 key selection box is applied in parallel to every 4-bit key-block. It performs the following boolean operation:

$$\begin{aligned} y(0) &= (x(0) \oplus x(1) \oplus x(2)) \cdot sel \vee (x(0) \oplus x(1)) \cdot \overline{sel}, \\ y(1) &= (x(1) \oplus x(2)) \cdot sel \vee x(1) \cdot \overline{sel}, \\ y(2) &= (x(2) \oplus x(3) \oplus x(0)) \cdot sel \vee (x(2) \oplus x(3)) \cdot \overline{sel}, \\ y(3) &= (x(3) \oplus x(0)) \cdot sel \vee x(3) \cdot \overline{sel}. \end{aligned}$$

Depending on the value of a selection bit *sel*, we obtain the round key RK_0^i or RK_1^i for round *i*.

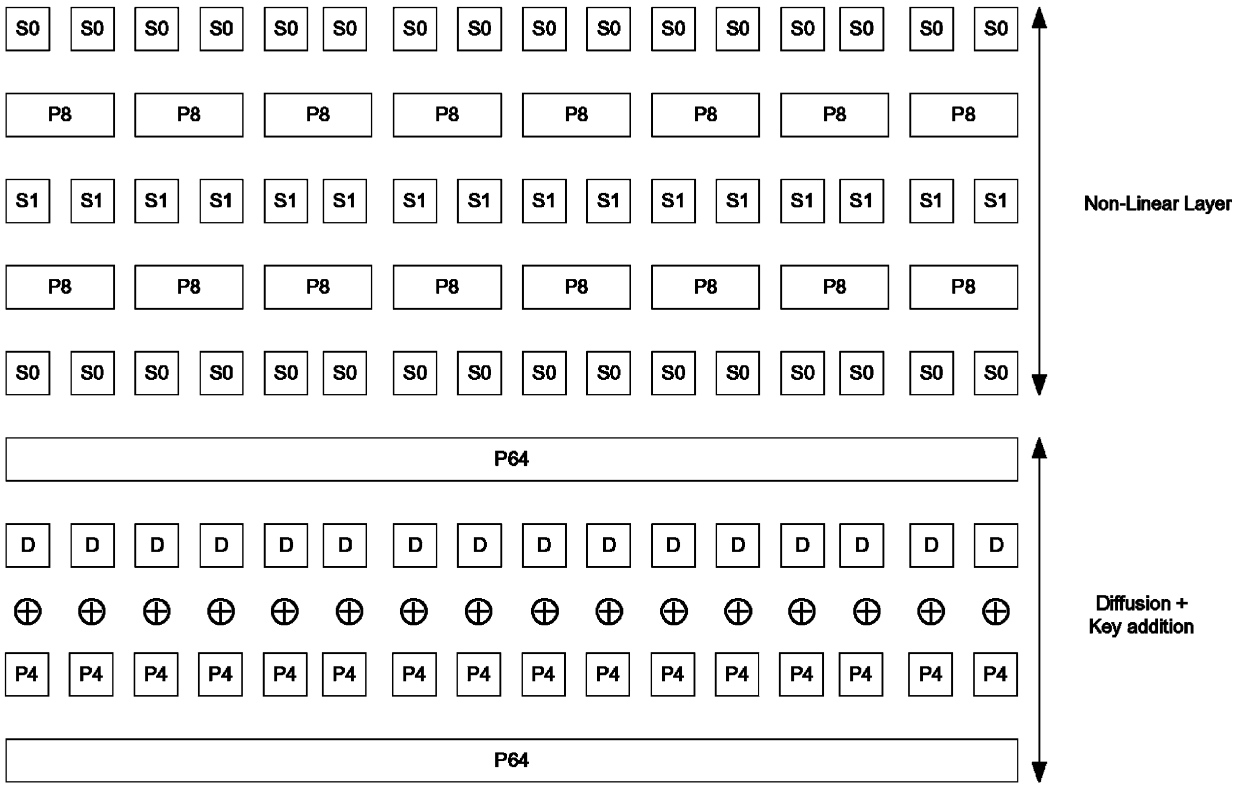

Fig. 1. The round function.