Chapter 11
Secure and Efficient Implementation of Symmetric Encryption Schemes using FPGAs

François-Xavier Standaert

11.1 Introduction

Due to its potential to greatly accelerate a wide variety of applications, reconfigurable computing has gained importance in the industrial development of digital signal processing systems. This chapter discusses how the particular properties of field programmable gate arrays (FPGAs) can be exploited for the secure and efficient implementation of symmetric cryptographic algorithms and protocols.

Reconfigurable computing intends to fill the gap between hardware and software, achieving potentially much higher performance than software, while maintaining a higher level of flexibility than hardware. Reconfigurable devices such as FPGAs contain arrays of computational elements whose functionality is determined through multiple programmable configuration bits. These elements, sometimes known as logic blocks, are connected using a set of routing resources that are also programmable. As a consequence, the realization of FPGA designs can be performed at the user site. Synthesis and implementation tools allow the high-level description of a hardware design to be translated into the programming file for an FPGA. The run-time operation of a reconfigurable system consequently occurs in two distinct phases: configuration and execution. First, the programming file of the reconfigurable device is directed from a host PC or an on-board memory to the FPGA. This configuration data are used to define the actual operation of the hardware. Thereafter, during the execution phase, the reconfigurable device acts as a purpose-built hardware.

The structure of actual computation blocks within the reconfigurable hardware varies from system to system. Each computation unit, or logic block, can be as simple as a 3-input function generator (usually denoted as look-up table (LUT)), or as complex as an 8-bit arithmetic and logic unit (ALU). This difference in the block size is commonly referred to as the granularity of the logic block. Fine-grain

UCL Crypto Group
e-mail: fstandae@uclouvain.be

Ç.K. Koç (ed.) *Cryptographic Engineering*, DOI 10.1007/978-0-387-71817-0_11,

blocks are useful for bit-level manipulations while coarse-grain blocks are better optimized for high-level data manipulations. The granularity of the FPGA also has a potential impact on the configuration time of the device. A fine-grained array has many configuration points to perform very small computations, and thus requires more data bits during reconfiguration. Recent FPGAs such as the one illustrated in Figure 11.1 usually combine different sizes or types of blocks in order to efficiently support different kinds of computations. For example, standard logic blocks using 4-input LUTs are combined with embedded RAM blocks, multipliers and microprocessors. Next to the computational blocks, the interconnections also have a major impact in the final performance of an FPGA. Recent devices are usually structured in different lengths of interconnects in order to efficiently deal with close and remote connections between the different logic blocks.

In the remainder of this chapter, we assume a reader with basic knowledge in FPGA design and cryptographic algorithms. Rather than providing a general introduction to reconfigurable cryptographic implementations, this chapter aims to put forward a number of properties of these devices and to discuss how they can be exploited efficiently and securely. Underlining how FPGA designs differ from standard integrated circuit designs with this respect is an alternative goal. Due to the very general nature of this topic, it is not intended to be extensively covered and our different sections attempt (as far as possible) to redirect the reader toward further readings when necessary. As introduction to the following issues, we suggest [10] for a general report on reconfigurable computing, [22] for detailed descriptions

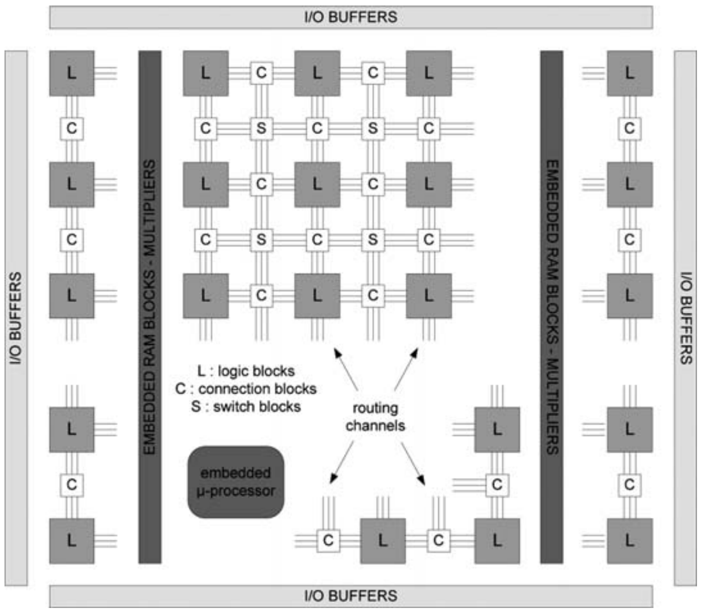

Fig. 11.1 High-level view of an FPGA.

on the efficient implementation of cryptographic algorithms on FPGAs, [11] for a good bibliography on FPGA security issues and [36] for a combined discussion of implementations and attacks on FPGAs. Additionally, since most of our running examples are using Xilinx FPGAs, we refer to [37] for a detailed description of these devices.

The rest of the chapter is structured as follows. Section 11.2 discusses the efficient exploitation of different FPGA features, from low-level facilities of the logic blocks to high-level architectural features. Section 11.3 details some metrics used for evaluating the efficiency of a cryptographic implementation. The two last sections investigate different issues related to the security of reconfigurable devices. Section 11.4 comments on the applicability of an important class of physical attacks (denoted as side-channel attacks) to recent FPGAs. Section 11.5 surveys other security topics, related to fault insertion and bitstream security. Our conclusions are in Section 11.6. We note finally that most of the examples used in this chapter are borrowed from implementation works of the UCL Crypto Group. Many similar results can be found in the literature and this chapter does not aim to give an overview of previously published works.

11.2 Efficient FPGA Implementations

This section considers the exploitation of different features in recent reconfigurable devices for the efficient implementation of symmetric cryptographic algorithms. We first investigate the slice structure, then describe the exploitation of additional embedded blocks and conclude by discussing the possible advantages of higher-level architectural facilities provided by recent FPGAs, namely embedded microprocessors and dynamic reconfiguration. We note that most of these features directly derive from the FPGA datasheets. However, since they are not always optimally (and automatically) exploited by the synthesis and implementation tools, it is important to have them in mind already during the high-level description of a cryptographic design.

11.2.1 Exploiting the Slice Structure

In this first section, we consider a Xilinx Virtex-II FPGA. Such devices embed programmable logic blocks, RAMs and multipliers. The slice is the logic unit that is generally used to evaluate an FPGA design's area requirements. Such a slice, depicted in Figure 11.3, is made up of two LUTs, two flip-flops (or registers) and a few additional gates. According to the user's choice, any of these LUTs can be configured in one out of three possible ways: RAM16 that acts like a $2^4 \times 1$-bit RAM storage, SRL16 that implements a 16-bit linear shift register and LUT that is capable of computing any 4-to-1 boolean function. A more detailed view of half a slice is given in Figure 11.2. An interesting thing to notice is the fast carry chain

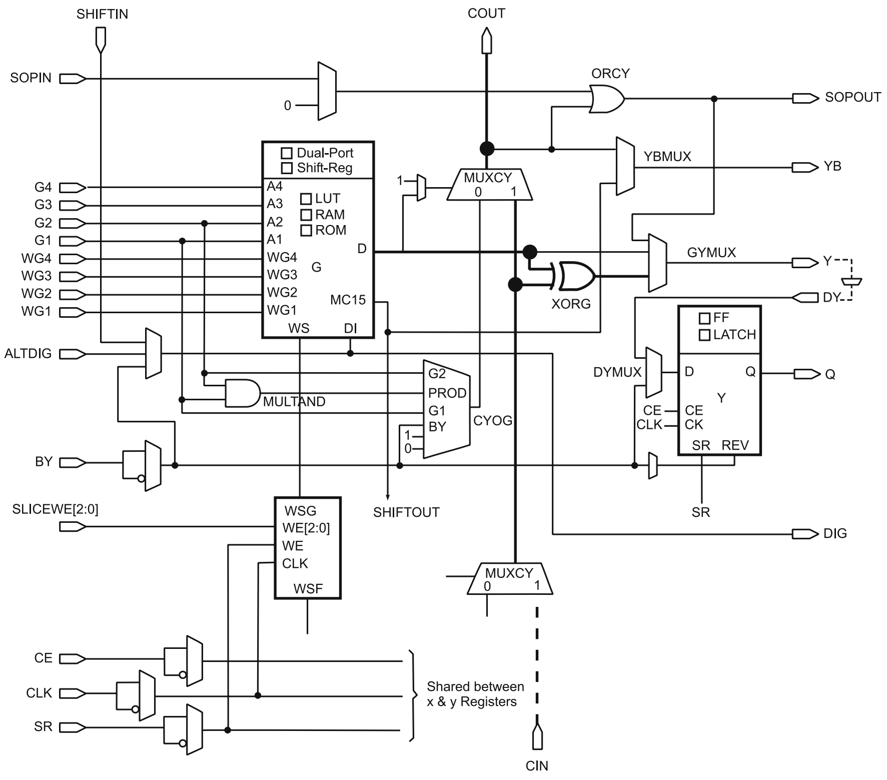

Fig. 11.2 Top half slice of a Xilinx Virtex-II.

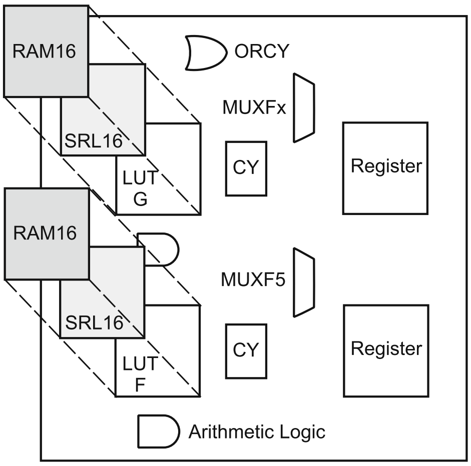

Fig. 11.3 Slice configurations.

(emphasized on the picture) crossing the slice. It allows the efficient implementation of carry propagate adders and, as will be shown, can sometimes be used to simplify the combinatorial cost of some designs.

11.2.1.1 The "Maximum" Pipeline Strategy and Limitations

The first observation from these pictures is that any LUT in a slice comes with its flip-flop. A consequence is that a straightforward pipelining strategy can be applied to the hardware design of, e.g., block ciphers in which the number of pipeline stages equals (or is close to) the design's logic depth expressed in LUTs. This is in contrast with ASICs where every flip-flop has its cost. Such strategies, looking for the maximum pipeline, generally give rise to very good synthesis results. However, the implementation (especially the routing task) of block ciphers within certain FPGAs can then become the bottleneck, due to high data diffusion. As mentioned, e.g., in [28], the design of a maximum pipelined advanced encryption standard (AES) Rijndael exhibits delays with 20% of logic and 80% of routes. It suggests that such a strategy is not optimum in these contexts. Improved solutions involve either the use of registers to pipeline the routes (if very high frequencies are to be reached) or the limitation of the design logic depth to two (or more) LUTs (if high efficiencies are required).

11.2.1.2 The Slice Multiplexors

Next to the slice LUTs, Xilinx FPGAs provide multiplexors (usually denoted as multiplexors Fx) allowing to efficiently implement distributed RAM and ROM within the FPGA. These elements have a strong impact on the implementation efficiency of encryption algorithms using substitution boxes, e.g., the AES Rijndael that uses a $2^8 \times 8$ S-box. As an illustration, the previously described Virtex-II allows implementing a $2^8 \times 1$ ROM with 16 LUTs and consequently, the AES Rijndael S-box fits in 64 slices. In the recent Virtex-5 family of FPGAs, the 4-input LUTs have been turned into 6-input LUTs. Using the same additional multiplexors allows to implement a $2^8 \times 1$ ROM with only 4 LUTs and consequently the AES Rijndael S-box fits into 32 slices.

11.2.1.3 The Shift-Register Slice Structure

A second convenient feature of the Virtex-II slice is the possibility to configure the LUT as a 16-bit shift register. This feature has a significant impact, e.g., in the implementation of stream ciphers using linear feedback shift registers. It is worth noticing that the efficient exploitation of the SRL16 primitives highly depends on the required taps positions in the stream ciphers. Any time an intermediate value is extracted from the SRL16 primitives, a new LUT has to be used. As a consequence, it may happen that a k-bit register with $k < 16$ occupies a complete SRL16 cell.

The stream cipher grain implementation in [7] is a good example of a straightforward (but very efficient) exploitation of these shift register-configured LUTs, with convenient taps positions.

Another interesting use of the shift register structure occurs when the target ciphers have an unbalanced structure. As a typical example, the data encryption standard (DES) has a very light key scheduling algorithm, compared to its round function. Therefore, the maximum pipeline strategy applied to the round and key round results in different number of pipeline stages. In a maximum pipeline implementation with "on-the-fly" round key derivation, several slices will consequently be "wasted" to pipeline the key schedule (meaning that their corresponding LUT will not be used). In such a context, the shift register structure can provide up to 16 pipeline stages with one single LUT, which result in a much more efficient implementation, e.g., in [23].

11.2.1.4 Additional Logic Gates Within the Slice

Finally, configurable logic blocks generally embed additional logic gates that can be efficiently exploited in certain specific contexts. One classical example is the XOR gate that is illustrated by the emphasized path in Figure 11.2. Since most symmetric encryption algorithms make an extensive use of such gates, they are generally useful to cryptographic designers. As an illustration, combining this gate with one LUT allows implementing a 5-bit XOR operation. In a maximum pipeline AES Rijndael implementation taking advantage of this 5-bit XOR, the combination of the MixColumns and AddRoundKey operations can consequently fit in only two pipeline stages, e.g., in [28]. Note that there is generally only one such XOR gate for several LUTs in a slice, which has to be taken into account during the designing phase (e.g., two LUTs can share the same XOR gate, but it has to have the same input).

11.2.2 Exploiting Embedded Blocks

Most recent FPGAs have an hybrid structure combining fine-grain logic blocks with larger-grain, specialized embedded blocks. Next to the inner structure of the FPGA logic blocks that was previously discussed, this section investigates how these larger blocks can be useful in the context of symmetric cryptographic implementations. For illustration, we selected two frequently available such blocks, namely embedded memories and multipliers.

11.2.2.1 RAM Blocks

Just as distributed RAM and ROM can implement the S-boxes of a block cipher, embedded memories can play the same role. However, in order to efficiently exploit these blocks, it is important to fill them as completely as possible, e.g., with the substitution tables defined in the target cipher specifications. As an illustration, the RAM blocks in the Virtex-E devices are dual-ported 4096-bit synchronous. Since the AES S-box has $2^8 \times 8 = 2048$ bits of memory requirements, it means that two S-boxes can fit in one such block. By contrast, Virtex-II devices incorporate dual-port synchronous RAM blocks of 18 Kbit. Storing the Rijndael S-boxes in such blocks is consequently not an efficient solution. An alternative proposal to exploit these larger memories is to implement both the S-boxes and the MixColumns operation as precomputed tables. Namely, let us consider the combination of SubBytes and MixColumns in Rijndael. An output column of this transform equals

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} SB(a_0) \\ SB(a_1) \\ SB(a_2) \\ SB(a_3) \end{bmatrix},$$

where the b_i's represent the combined transform output bytes and the a_i's its input bytes to the S-boxes. Therefore, if we define four tables as

$$T_0(a) = \begin{bmatrix} 02 \times SB(a) \\ SB(a) \\ SB(a) \\ 03 \times SB(a) \end{bmatrix}, \quad T_1(a) = \begin{bmatrix} 03 \times SB(a) \\ 02 \times SB(a) \\ SB(a) \\ SB(a) \end{bmatrix},$$
$$T_2(a) = \begin{bmatrix} SB(a) \\ 03 \times SB(a) \\ 02 \times SB(a) \\ SB(a) \end{bmatrix}, \quad T_3(a) = \begin{bmatrix} SB(a) \\ SB(a) \\ 03 \times SB(a) \\ 02 \times SB(a) \end{bmatrix},$$

the combination of SubBytes and MixColumns equals

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = T_0(a) \oplus T_1(a) \oplus T_2(a) \oplus T_3(a).$$

The size of one T_i table is $2^8 \times 32 \simeq 8$ Kbits. It is consequently possible to store the four tables in two dual-port 18 Kbit RAM blocks, e.g., as in [24]. Similarly, such 18 Kbit memory blocks offer a straightforward solution to implement a masked DES design to improve the security against side-channel attacks (see Section 4.2 for details). As the DES S-boxes have $2^6 \times 4$ bits of memory requirements, its masked counterpart has $2^{12} \times 4 \simeq 16$ Kbit of memory requirements, which just fits into the Virtex-II RAM blocks [32]. It is finally important to note that due to their fixed position within the device, embedded RAM blocks impose stronger routing constraints than their distributed counterpart of the previous section. It may affect the operating frequency of a design, specially if a large number of embedded blocks are used.

11.2.2.2 Embedded Multipliers

Although arithmetic operands are more likely to be exploited in asymmetric cryptographic applications, there are examples of symmetric ciphers in which one can also take advantage of embedded multipliers. In the IDEA block cipher, integer multiplication modulo (2^n+1) is usually the bottleneck for hardware implementations. Such constraints suit pretty well to FPGAs embedding small multiplier blocks, such as the Virtex-II family of devices [4].

11.2.3 Exploiting Further Features

The previously described FPGA features related to the logic block structure and the use of embedded memories and multipliers have been intensively used in a variety of implementation works. In this section, we briefly describe some more recent (and less-investigated) trends in the design of reconfigurable systems that can potentially be exploited in cryptographic applications.

11.2.3.1 Microprocessors and Controllers

Microprocessors and controllers of two shapes can be found in recent reconfigurable devices. First, hard cores can be discretely embedded in the device, as the previously described RAM blocks and multipliers. This is typically the case of the PowerPC microprocessor that is available in recent Xilinx devices, e.g., the Virtex-II-pro. Second, micro-controllers can be synthesized and implemented within the FPGA logic blocks, just as distributed memories. This is typically the case of the MicroBlaze (32-bit RISC) and PicoBlaze (8-bit RISC) controllers that are freely available as soft cores from Xilinx.[1] In a processor-based FPGA system, customized IP cores can then be connected to the controllers through various interfaces like the on-chip-peripheral bus (OPB) or the fast simplex link (FSL). In general, processor-based embedded systems cannot be compared favorably with specialized cryptographic designs in which the hardware is optimized and possibly pipelined in order to reach high implementation efficiencies. However, when the system specifications establish that a general-purpose processor must be used, the MicroBlaze and PicoBlaze solutions can be suitable. They additionally offer the flexibility of being programmed with a software language. For example, [13] describes the implementation of various block ciphers within a MicroBlaze-based system.

11.2.3.2 Dynamic Reconfiguration

Dynamic reconfiguration refers to the possibility of reconfiguring an FPGA partially, while operating and without compromising the integrity of the application

[1] As an illustration, a PicoBlaze core takes less than 200 logic cells in a Spartan-II device and can run at 76 MHz. A MicroBlaze core takes less than 1000 logic cells in a Virtex-II device and can run at 125 MHz.

running. It is sometimes referred to as partial or run-time reconfiguration [18]. From a theoretical point of view, dynamic reconfiguration allows using more hardware than physically present in the FPGA which can be used to reduce the size of the device as well as its overall power consumption. From an application point of view, the expected benefits include any adaptive change of the FPGA design due to environmental changes (e.g., a change of algorithm or change of performance constraints). However, the exploitation of such techniques pose a number of practical issues, including the reliability of the design flow and the time required for the reconfiguration. Its exploitation in cryptographic applications is therefore a scope for further research, although it appears as a promising opportunity to improve systems flexibility.

11.2.4 Combining the Tricks: The Flexibility Versus Efficiency Tradeoff

To conclude this section, let us first mention that all the previous tricks only constitute a part of the possibilities offered by recent FPGAs, can be efficiently combined and generally have to be considered during the high-level modeling stage of a hardware design. For example, the way the inner structure of the slice can be exploited strongly determines the pipelining strategy to use. Second, it is important to consider that the optimal exploitation of one specific target FPGA, by designing in function of the slice structure or available embedded blocks, makes the hardware code less portable. It also sometimes requires to map some parts of the design by hand into the FPGA resources. There is consequently a tradeoff to find between the efficient exploitation of a given device and the possibility to use an IP core in a variety of systems and products. Note finally that although our illustrative implementation examples are based on symmetric cryptographic algorithms, the techniques discussed in this section generally apply for any reconfigurable hardware design.

11.3 Fair Evaluation of a Cryptographic FPGA Design

Before any cryptographic design is implemented always comes the question of the performance goals to achieve. Stating these goals properly in function of a target application and determining good metrics for the performance evaluation is therefore an important step in the understanding of reconfigurable architectures. Unfortunately, there is no straightforward answer to these questions and the fair evaluation (or comparison) of a given FPGA implementation is often a matter of taste. This section aims to illustrate some important questions to consider in such a performance assessment. For this purpose, we restrict ourselves to the implementation of the AES Rijndael. We start by considering the design goals. Then we discuss the performance evaluation.

11.3.1 Design Goals

A list of design goals for the FPGA implementation of the AES Rijndael would typically include (but is not limited to) the following eight questions:

1. Does the application require to develop an encryption/decryption core or just an encryption only, decryption only core?
2. Is the key scheduling algorithm required to be performed "on-the-fly" or can the round keys be computed once and stored in memory?
3. Is the block cipher design supposed to run in a specific encryption mode (e.g., feedback) that would prevent the use of pipelining?
4. What kind of interface has to be provided to the outside world?
5. Are there specific constraints to be fulfilled by the implementation (e.g., in terms of hardware cost or throughput)?
6. What is the target FPGA device? With which speedgrade?
7. Are there available embedded blocks in the device (are not they required for running other applications than cryptographic ones)?
8. What is the datapath size planned for the design (128-bit, 32-bit, etc.)?
9. Are multiple clocks allowed within the reconfigurable system?

11.3.2 Performance Evaluation

Assuming a hardware designer has implemented the AES Rijndael following some of the design goals in the previous section, its performances could then be measured with the following metrics: hardware cost (in LUTs, registers, slices, etc.), operating frequency (in MHz), throughput (in Mbit/sec) and possibly some efficiency measurement, e.g., throughput/hardware cost. For illustration, Table 11.1 lists some exemplary AES Rijndael implementations with selected design goals and Table 11.2 summarizes their performances according to selected metrics. These tables typically illustrate the difficulty of performing fair comparisons between different FPGA designs. First, different architectures generally have different design goals. Second, evaluation metrics can be misleading since they highly depend on the target device. Comparing the performances of different algorithms raises similar questions.

Table 11.1 Exemplary AES Rijndael designs with selected design goals.

Index	E,D	Key Sched.	Feedback	Device	Architecture
1.	E only	On-the-fly	no	Virtex-E	128-bit unrolled
2.	E only	On-the-fly	no	Virtex-E	128-bit loop
3.	E/D	Precomputed	yes	Virtex-II	32-bit loop
4.	E/D	Precomputed	yes	Spartan-II	8-bit loop
5.	E/D	Precomputed	yes	Spartan-II	PicoBlaze

Table 11.2 Exemplary AES Rijndael designs with selected performance metrics.

Index	Ref.	LUTs	Regs.	Slices	RAMBs	Freq.	Throughput
1.	[28]	3516	3840	2784	100	92 MHz	11.7 Gbit/sec
2.	[28]	3846	2517	2257	0	169 MHz	2 Gbit/sec
3.	[24]	288	113	146	3	123 MHz	358 Mbit/sec
4.	[14]	–	–	124	2	67 MHz	2.2 Mbit/sec
5.	[14]	–	–	119	2	90 MHz	710 Kbit/sec

Note that efficiency metrics (e.g., throughput/hardware cost) can be specially misleading since the hardware cost in FPGAs can be expressed in LUTs, slices, RAMBs, etc. Some metrics consequently attempt to unify these different resources, e.g., by expressing the cost of the RAM blocks as distributed RAMS in LUTs, but this is still device dependent. General observations can nevertheless be highlighted. For example, looking at the dependencies between the architecture size and the throughput in the previous tables, one could state that applications in the multi-Gbit/sec range should consider 128-bit unrolled architectures, applications in the Gbit/sec range should consider 128-bit loop architectures, applications in the hundreds of Mbit/sec range should consider 32-bit loop architectures and so on. As previously mentioned, these tables are far from being a complete survey of existing implementations of the AES Rijndael nor do they contain the best available results. For a more detailed list of such implementations, we refer to [15].

11.4 Security of FPGAs Against Side-Channel Attacks

The previous sections mainly cared about efficient FPGA implementations. However, as far as cryptographic algorithms are concerned, not only their hardware cost, throughput, etc. are important to a designer but also their security against various types of physical attacks. Physical attacks on cryptographic devices take advantage of implementation-specific characteristics to recover the secret parameters involved in the computations. They are therefore much less general – since specific to a given implementation – but often much more powerful than classical cryptanalysis and are considered very seriously by cryptographic device manufacturers. Examples of physical attacks include the probing of devices [2], the insertion of faults [5] or the monitoring of side-channel information leakages such as the power consumption [17] or electromagnetic radiation [1]. Due to the important amount of public work that has been dedicated to the analysis of side-channel attacks against FPGAs, this section discusses their specificities. In Section 11.5, we consider other aspects related to the tamper resistance of reconfigurable devices, including fault attacks and bitstream security issues. As for the previously discussed efficiency concerns, we do not aim to present an exhaustive survey of physical attacks on FPGAs but to put forward a number of their meaningful features. We redirect the reader toward further readings when needed.

11.4.1 Applicability of the Attack and FPGA Properties

Side-channel attacks are based on the hypothesis that an exploitable amount of secret information is leaked by an implementation through a physical channel. For example, in power analysis attacks, an attacker uses a hypothetical model of the device under attack to predict its power consumption. These predictions are then compared to the real measured power consumption in order to recover secret information (i.e., secret key bits of block ciphers). In this first section, we aim to illustrate that such physical information is indeed leaked by FPGA devices and can be exploited, using simple attack models. For this purposes, we focus on static RAM-based reconfigurable devices (like the previously considered Xilinx Virtex family) since they are the most popular technology in use. In these devices, the storage cells, the logic blocks and the connection blocks are made of CMOS gates.

11.4.1.1 A Simple Leakage Model Applicable to FPGAs

Static CMOS gates have three distinct dissipation sources [21]. The first one is due to the leakage currents in transistors. The second one is due to the so-called short-circuit currents there exists a short period during the switching of a gate while NMOS and PMOS transistors are conducting simultaneously. Finally, the dynamic power consumption is due to the charge and discharge of the load capacitance C_L represented by the dotted paths in Figure 11.4. The respective importance of these dissipation sources typically depends on technology scalings. But the dynamic power consumption is particularly relevant from a side-channel point of view since it determines a simple relationship between a device's internal data and its externally observable power consumption. It can be written as

$$P_{dyn} = C_L V_{DD}^2 P_{0\rightarrow 1} f, \tag{11.1}$$

where $P_{0\rightarrow 1}$ is the probability of a $0 \rightarrow 1$ bit transition, f is the operating frequency of the device and V_{DD} is the voltage of the power supply. Therefore, in practice, a

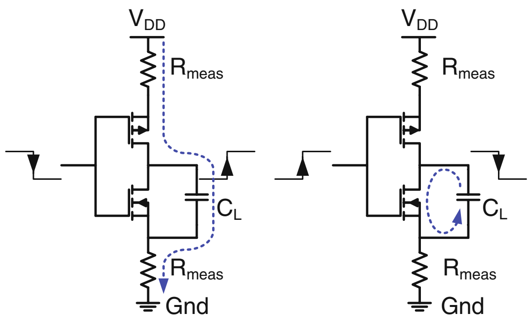

Fig. 11.4 Charge versus discharge of a CMOS inverter.

simple way to model the power consumption of an FPGA is to predict its switching activity. Let S_1 be a large bit register containing the state of all the FPGA cells at some moment in time t_1 and S_2 be another register containing the FPGA state one clock cycle later. The number of bit switches (including both $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions) in the device when moving from state S_1 to S_2 equals the Hamming distance between these states, namely $H_D(S_1,S_2) = H_W(S_1 \oplus S_2)$, where H_W is the Hamming weight operator.

11.4.1.2 Exploiting the Leakages

We illustrate the attack principle with the simple encryption network of Figure 11.5, which contains the same basic elements as most present block ciphers e.g., the AES Rijndael. That is, the plaintext is XORed with a secret key, then goes through a layer of relatively small substitution boxes and is finally sent to a larger permutation (e.g., a linear diffusion layer for the AES Rijndael). The same operations are iterated a number of times. For the purposes of this chapter, it is not necessary to know more details on these algorithms. The attack proceeds as follows. Let the adversary target the 4 key bits entering the left S-box of Figure 11.1, denoted as $K_0[0...3]$. Then, for N different plaintexts, he first predicts the number of transitions at the S-box output, for every possible value of $K_0[0...3]$. The result of this prediction is a $N \times 2^4$ prediction matrix $\mathbf{P}$, containing numbers between 0 and 4. In the second part of the attack, the adversary lets the circuit encrypt the same N plaintexts with a fixed secret key and he measures the power consumption of the device while the chip is operating the targeted operation. For each plaintext, he stores a single value for the power consumption (e.g., the average or maximum value of the target clock cycle). This results in a $N \times 1$ measurement vector $\mathbf{M}$. Finally, the attacker computes the correlation[2] between the measurement vector and all the columns of the prediction

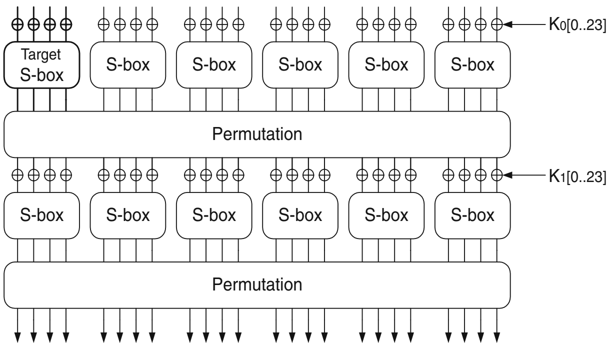

Fig. 11.5 A simple encryption network.

2
$$C(\mathbf{M},\mathbf{P}) = \frac{\mu(\mathbf{M}.\mathbf{P}) - \mu(\mathbf{M}).\mu(\mathbf{P})}{\sqrt{\sigma^2(\mathbf{M}).\sigma^2(\mathbf{P})}}, \tag{11.2}$$

where $\mu(\mathbf{M})$ denotes the mean of the set of measurements and $\sigma^2(\mathbf{M})$ its variance.

matrix (corresponding to all the possible key guesses). If the attack is successful, it is expected that only one value, corresponding to the correct key bits, leads to a high correlation.

Such attacks have been successfully applied to different algorithms implemented on a variety of FPGA devices. For example, an attack against the simple design of Figure 11.5 has been implemented against a Xilinx Spartan-II device and its results are illustrated in Figure 11.6 in which the correct key candidate is clearly distinguishable. We note that different statistical tools could be considered to mount power analysis attacks and the use of the correlation coefficient is not optimal with this respect. For example, maximum likelihood techniques [9] may yield better results. However, with the simple power consumption models considered here, correlation attacks provide good results and are extremely easy to manipulate (e.g., they do not require any estimation of the noise in the target devices). Note finally that the same set of measurements can be used to recover all parts of the key, by changing the prediction matrix (i.e., by applying a divide and conquer strategy).

11.4.1.3 Exemplary FPGA Properties

To summarize the previous paragraphs, there are two important aspects to take into account in the analysis of side-channel attacks. First, the target implementation has to leak some information. With this respect, recent FPGAs are made of CMOS

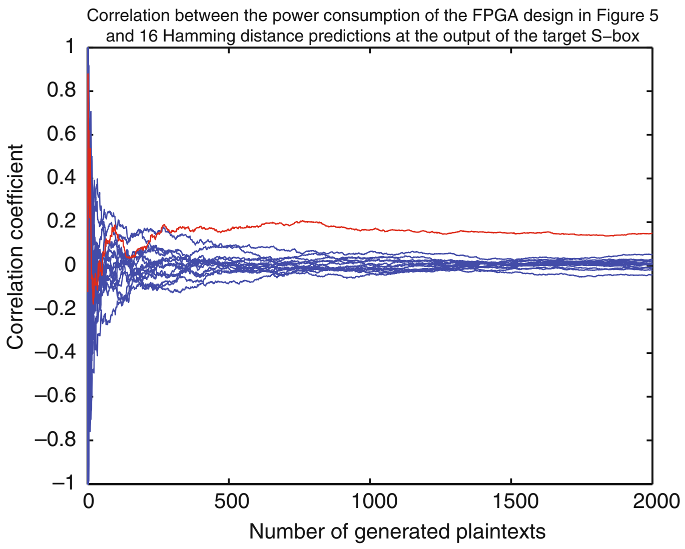

Fig. 11.6 Exploitation of the side-channel leakages.

transistors, just as smart cards and controllers. They consequently leak information, just as smart cards and controllers. Second, this information has to be exploitable by the adversary. With this respect, FPGA implementations offer opportunities to perform parallel computing and are based on a specific array of logic and routing cells. These features (among others) lead to specific approaches in the exploitation of side-channel leakages.

- *Parallel computing and target leakage*. In power analysis attacks, the leakage provides an adversary with an image of the computation performed within a target device. Depending on the implementation context (e.g., see the different architectures in Table 11.1), this information relates to 8-bit, 32-bit, 128-bit (or more) computations. But looking at Figure 11.5, a side-channel adversary typically targets small parts of the computation one by one, corresponding to small (e.g., 4-bit) parts of the key. Therefore, the power consumption due to the untargeted parts of the computation generates what is usually denoted as algorithmic noise. Compared to smart cards and controllers, FPGAs offer the specific opportunity to design large architectures, with a significant amount of such noise.
- *FPGA structures and leakage models.* Even more specific of FPGAs is the array structure of Figure 11.1. In this structure, the different computational elements are connected through different types of wires. A consequence is that the different bits in an implementation contribute differently to the overall power consumption, due to different effective capacitances. These different capacitances have been highlighted, e.g., in [26] for the Virtex-II family of devices. A consequence is that the simple Hamming distance model for the prediction of the power consumption in reconfigurable devices can be improved according to these effective capacitances, e.g., by assigning different weights to the switches of different bits within a design. As an illustration, the left part of Figure 11.7 depicts

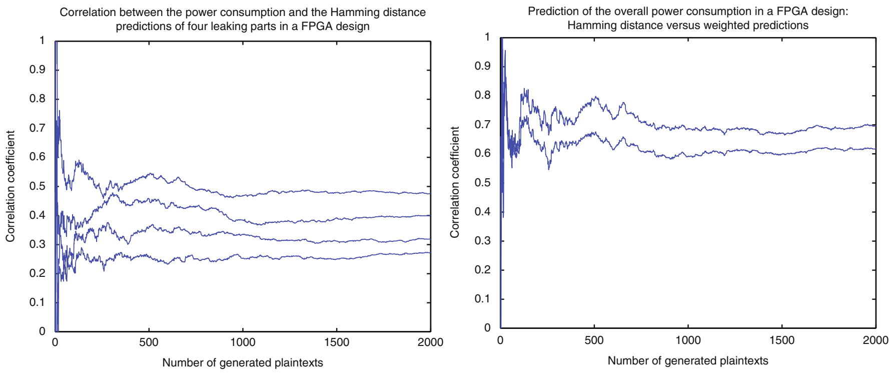

Fig. 11.7 Resource-dependent correlations in Xilinx FPGAs.

the correlation of four parts of a computation in an FPGA and their contribution to the overall power consumption (measured with a correlation coefficient). The right part of the figure illustrates that assigning different weights to these computations allows a better prediction of the overall power consumption [31], which results in a more efficient side-channel attack.

To conclude this section, the state-of-the-art side-channel attacks against FPGA devices, e.g., as surveyed in [30], typically illustrate the following three facts: (1) Side-channel are a threat for FPGAs, as for any microelectronic cryptographic device. (2) Just as FPGAs offer nice opportunities for efficient implementations, they have interesting features for secure implementations as well (e.g., the parallel computing opportunity or the ability to design datapaths including the countermeasures of the following section). (3) Just as the optimal exploitation of the FPGA structure is useful for efficiency (as detailed in Section 11.2), exploiting the architectural properties of a given target device (e.g., the effective capacitance of the different computational parts in a design) is useful for improving the efficiency of an attack.

11.4.2 Countermeasures

Countermeasures against side-channel attacks range among a large variety of solutions. However, in the present state of the art, no single technique allows to provide perfect security. Protecting implementations against physical attacks consequently intends to make the adversary's task harder. In this context, the implementation cost of a countermeasure is of primary importance and must be evaluated with respect to the additional security obtained. The exhaustive list of all possible solutions to protect cryptographic devices from side-channel opponents would deserve a long survey in itself. In the following, we list four illustrative solutions to improve the resistance against power analysis that are applicable to FPGAs. Obtaining practical security usually requires to combine them (possibly with others) in a clever way.

- *Noise addition.* Adding noise to the side-channel measurements is a very common technique to reduce the amount of information in the leakages. This can be achieved in a variety of ways, at different abstraction levels, e.g., physical, technological, algorithmic. As previously mentioned, the use of large architectures producing a significant amount of algorithmic noise is interesting with this respect and easy to apply to FPGA designs [29].
- *Data randomizations* intend to make all the cryptographic computations within the FPGA dependent on some unknown random values generated on-chip. It makes the prediction of the power consumption more difficult. Masking is a typical example of such countermeasures that has been intensively studied in the literature and applied to FPGAs, e.g., in [20, 32].
- *Random pre-charges* are another solution to make the side-channel leakage harder to exploit. If one every two inputs to an encryption design is a random number generated on chip, an adversary will not be able to predict the transitions

within the implementation anymore (of course, the random ciphertexts should not be outputted from the device). As suggested in [31], a solution for the adversary is then to distinguish between $0 \rightarrow 1$ and $1 \rightarrow 0$ bit transitions in the leakages. But it results in worse leakage models and less-efficient attacks than in the un-protected Hamming distance model.

- *Dynamic and differential logic styles* finally intend to make the power consumption within the FPGA independent of the computed data. A logic style is denoted as differential if the complementary data inputs and outputs are available in the circuit. The notion of dynamic logic gates refers to the fact that the gate operation is divided into two phases [21]. First, the output capacitance is charged. Then, during the evaluation, it is discharged according to the input values. When combining dynamic and differential logic styles, there are always two capacitances loaded during the pre-charge and one of them is discharged during the evaluation, regardless of the input sequences. In [34], such a circuit behavior is proposed for FPGAs.

11.4.3 Measuring Side-Channel Resistance

Countermeasures against side-channel attacks as listed in the previous section usually involve a significant performance overhead for the encryption algorithms. Therefore, just as hardware efficiency is a design goal that has to be evaluated with (hopefully) fair metrics, physical security also has to be evaluated properly. In this section, we briefly refer to the proposed evaluation methodology introduced in [33] for these purposes. We use the intuitive picture of Figure 11.8 in which side-channel attacks are viewed as a communication problem. In summary, there are two important aspects to consider in the analysis of side-channel attacks. First, the amount of information leaked by a target device can be measured with the conditional entropy (or mutual information). Second, the extent to which an adversary can exploit this information can be measured with its success rate, just as the bit-error-rate does in

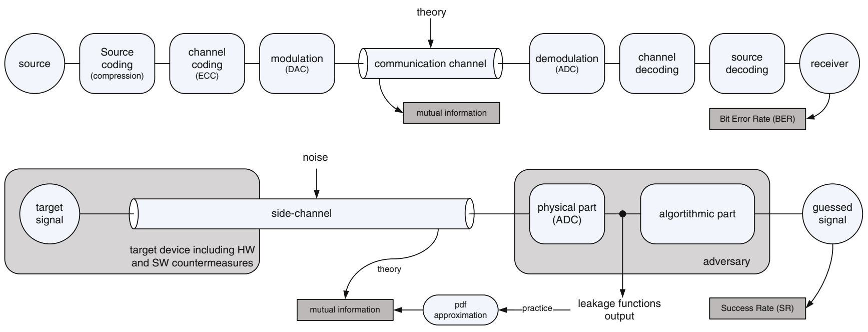

Fig. 11.8 Digital communications versus side-channel attacks.

communication problems. By combining both measurements, one can evaluate both the quality of an implementation and the strength of a side-channel adversary. Trading efficiency for security consequently requires to evaluate how the addition of a countermeasure in a design affects these two metrics.

11.5 Other Security Issues

Side-channels attacks are only a part of the concerns related to the implementation of cryptographic algorithms within reconfigurable devices. *Readback attacks, cloning of the devices, bitstream reverse-engineering and fault attacks* are other concerns that could be considered [36]. In this section, we review some issues related to the insertion of faults in FPGAs and to their bitstream security. They both constitute research challenges for future applications.

11.5.1 Fault Attacks

Similarly to side-channel attacks, fault attacks are an intensively studied adversarial model for cryptographic implementations. However, only a small number of experiments can be found in the literature on the actual possibility to apply such attacks to FPGAs. Preliminary results as in [8] suggest that fault insertion is feasible (as for any other SRAM-based device), but could be more difficult to exploit than in the context of smaller devices like smart cards. The large number of memory cells present in the logic arrays, determining at the same time the computational state of a cryptographic algorithm and the configuration of the device (including logic and routing), implies that different types of faults can occur. “How efficiently can these faults be exploited?” or “Can they hurt the FPGAs permanently?” are exemplary open questions. As for side-channel attacks, it is reasonable to expect that security against such attacks will require to add countermeasures (some of them surveyed in [19] for block ciphers) and therefore to trade some of a design’s efficiency for security.

11.5.2 Bitstream Security

Bitstream security is a critical issue for SRAM-based FPGAs. The recovery of bitstreams (e.g., by applying readback attacks in which the configuration file is read out of the FPGA) in order to clone an FPGA or their reverse-engineering and also the digital rights management (DRM) of the intellectual property (IP) cores are important concerns for the electronic industry. In this section, we survey some of these questions and describe the (partial) solutions that have been proposed by the industry. We start with a (simplified) description of the different parties in the game. Then, we discuss the interactions between these parties and the related security problems.

11.5.2.1 Parties in the FPGA Business

In order to keep our descriptions as simple and straightforward as possible, we limit our discussions to a three-player game, namely the "*end user*", the "*system designer*" and the "*IP provider*". A more detailed description of the FPGA IP transactions can be found in [16]. The IP provider delivers the hardware description language (HDL) files (or any other suitable file format such as the netlist for a particular FPGA) for some specific algorithms, e.g., encryption, image processing. The system designer creates a complete design for an FPGA chip, making use of one or more IP cores purchased from IP providers, e.g., a hardware decoder for the digital cinema [25]. Finally, the end user takes advantage of equipment containing FPGAs.

11.5.2.2 Bitstream Security: System Designer Versus End User

In this interaction, the main goal for the system designer is to prevent readback attacks, cloning of the FPGAs and reverse-engineering of the bitstream. Otherwise said, the FPGA should appear as black box to the end user. Since the bitstream is generally stored in an EPROM, an additional issue is to securely connect this external memory and the FPGA. For all these purposes, the most frequently considered solution is the bitstream encryption illustrated in Figure 11.9. In this solution, the bitstream is encrypted by the CAD tool with user-defined symmetric (secret) keys. The same keys are stored on the FPGA, e.g., in a volatile memory with an external battery. During configuration, an on-chip decryption circuit is used to recover the original configuration file. Readback is not allowed when encrypted bitstreams are used.

Although this or similar methods are used in several commercial devices, they suffer from a number of drawbacks. First, it requires an external battery to store the key. Second, it requires an on-chip decryption circuitry. But most importantly,

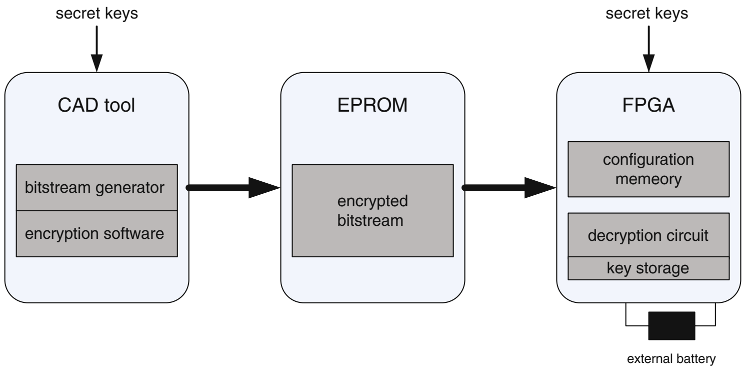

Fig. 11.9 Bitstream encryption in Virtex-II FPGAs.

the key management of such a solution is tricky. For example, if a single key is used for all the boards, then a system designer has no opportunity to update the configuration files for only a part of them. Ideally, it should be possible to update the symmetric keys remotely. This could be achieved either by the use of a symmetric master key (but the system security would then depend on this single key) or a public key mechanism in which each FPGA would come with a private/public key pair stored in a non-volatile memory.

11.5.2.3 IP Cores DRM: IP Provider Versus System Designer

In this interaction, the situation is even more difficult. First, the IP provider does not have the FPGA devices in hand which prevents him to store symmetric keys as in the previous section. Second, the system designer would like to be able to integrate the IPs in a larger design and to simulate it.

In present devices, a solution for the IP provider to deal with these issues is to send the system designer an encrypted netlist and simulation model. Xilinx development tools allow dealing with such files by embedding a secret key in its software. But the security of this solution entirely relies on this single key which may consequently be the target of reverse-engineering attempts. In addition, this model only allows a "per IP core license" business model.

A proposal to allow a "per device license" business model is described in [38]. For this purpose, and for every unit to be built, the IP provider feeds the system designer with both an IP core and a pre-programmed external security chip. A secret key (chosen by the IP provider) is stored in the encrypted netlist of the IP and the same key is embedded in the security chip. Before the IP can run on a board, it checks that the security chip embedding the correct key is properly connected to the FPGA. The hardware to do this security check is part of the IP core. This solution allows the IP provider to monitor the number of devices running its design but suffers from several drawbacks. First, the key management is not easy since the key is embedded in the netlist. As for the bitfile encryption, key updates could be made easier by using either a master key or a public key mechanism in the FPGAs. Second, the system security does still rely on netlist encryption in the Xilinx software. If an adversary can decrypt the netlist, he can also disable the security check.

There is consequently a tradeoff to face in the DRM of IP cores. For flexibility reasons, it is desirable that the security relates to the netlists so that IPs can be easily simulated and integrated in larger designs (as in the previous proposal). But for security reasons, the best solution would be to deal directly with bitstreams. For example, if a non-volatile private key K_s and the corresponding public key K_p were available in an FPGA, an IP provider could sell a pair $[E_K(\text{bitstream}), E_{K_p}(K)]$ to protect its design, in a "per device license" business model. But present development tools do not allow to easily combine different bitstreams which makes this solution quite unpractical for the system designer's point of view. Improved solutions (e.g., taking advantage of partial reconfiguration techniques) are consequently required to improve this setting. Note finally that, whatever the DRM and bitstream security mechanisms involved, the underlying cryptographic algorithms may still be the target of side-channel or fault attacks. It is therefore important to quantify the reliability of these solutions with an appropriate security level.

11.6 Conclusions and Open Questions

This chapter discussed some aspects in the secure and efficient implementation of symmetric encryption schemes in recent FPGAs. It aims to illustrate both how the particular properties of these reconfigurable devices can be exploited to improve the performances of an implementation and how the same properties can be exploited by malicious adversaries. Our discussions suggest different tradeoffs for cryptographic designers. First, the flexibility of a design can be traded for performances. That is, by carefully taking advantage of all the architectural details of a given device, one can improve performance at the cost of a less-portable hardware code. Second, the performances of a design can be traded for physical security. That is, resisting against fault or side-channel attacks usually involves overheads in the design efficiency.

From a technological point of view, open questions in the field relate to the effect of technology scalings in the future generations of FPGAs, both in terms of performances and security against physical adversaries. From an application point of view, and as the capacity of FPGAs increases to millions of equivalent gates, the protection of IP cores with secure DRM solutions becomes increasingly important. The development of IP protection schemes that do not harm the flexibility of the development tools is therefore an important requirement. It should allow IP providers, FPGA system designers and end users to interact in a fair and secure business model. The integration of a public key mechanism by FPGA manufacturers or the exploitation of physically unclonable functions within FPGAs, e.g., as suggested in [27, 35] appear as promising approaches with this respect.

Acknowledgements The author would like to thank Fraçois Macé, Guerric Meurice and Gaël Rouvroy for meaningful comments on this work.

11.7 Exercises

Khazad [3] is an iterated 64-bit block cipher with 128-bit keys. It comprises eight rounds; each round consists of eight 8-bit to 8-bit S-box parallel look-ups, a linear transformation (multiplication by a constant MDS diffusion matrix) and round key addition. For efficient hardware implementations, the 8-bit to 8-bit substitution is made of six smaller 4-bit to 4-bit substitutions.

1. Assume an FPGA with 4-bit LUTs and dual-ported 4096-bit synchronous RAM blocks. What is the cost of the complete Khazad substitution layer in LUTs and RAMBs? What are the respective memory requirements (in bits) of the LUT-based and RAMB-based solutions for the S-box?
2. Consider the 64-bit loop architecture for a (simplified) round of Khazad in Figure 11.7. Assume that each layer (NL1, NL2, NL3, L) has a cost of 64 LUTs (fully utilized). What is the total cost of such a design in LUTs (including the key addition and the input multiplexor)?
3. Assume a very simple key scheduling that can be implemented as a single layer of 128 LUTs. How much LUTs and registers are required to pipeline such a key scheduling in five levels in the best manner if a slice structure similar to the one of Figure 11.3 is used?
4. Assume that the delay of a LUT equals 5 nsec and that only these delays determine the operating frequency of the design. What maximum throughput can be obtained with a 2 (*resp*. 5) pipeline stage strategy if eight rounds have to be iterated (in Mbits/sec)?
5. Same question if the delay of a LUT equals 3 nsec but there is a fixed delay due to routing constraints in the design of 10 nsec. What is the best throughput that can be obtained in a feedback mode in this context?
6. Consider a side-channel adversary trying to recover a n-bit random value k who obtains the Hamming weight of this value: $H_W(k)$. Assuming noiseless measurements, how much information does he gain? Now assume $n = 64$ as for the Khazad cipher and an adversary who would obtain the Hamming weight of first round S-box layer's output for different plaintexts. Can an adversary exploit all the information on k in a correlation attack? (hint: think both about information and computation).

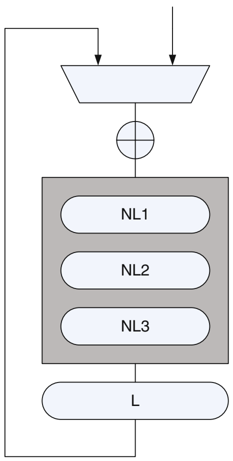

Fig. 11.10 Loop architecture for a (simplified) round of Khazad.

7. Consider a side-channel adversary targeting part of the key in the design of Figure 11.7. Assume that the NL layers do not provide cryptographic diffusion and that the L layer does provide perfect diffusion. Assume a five pipeline stage implementation in which the overall power consumption of the design is only caused by the registers. Assume that the correlation between the predictions of the adversary and the real measurements (denoted as ρ) equals the square root of the number of predicted registers in an attack divided by the total number of registers in the design. Assume finally that the number of plaintexts required for a successful attack can be approximated with $N_{succ} \simeq c \cdot \frac{1}{\rho^2}$. How many plaintexts are required for a successful attack against an 8-bit part of the key for which $c = 10$?
8. Consider now an implementation in which two rounds of Khazad are unrolled. How much would the security against the previous side-channel adversary be increased? Consider finally the same implementation protected by a countermeasure such that the correlation coefficient is reduced by a factor of 5. What number of plaintexts would then be required to attack?
9. Evaluate the hardware cost and throughput of the previous two-round unrolled implementation of Khazad ($\Delta_{\text{lut}} = 5$ nsec). Then assume that the previous countermeasure against side-channel attacks (reducing the correlation coefficient by a factor of 5) is applied to a single round architecture, divides the throughput by two and uses 250 additional LUTs in the design (e.g., it could be a design with random pre-charges). Can you comment the efficiency versus security against correlation attacks tradeoff for these designs? Which metrics can be used for these purposes?

11.8 Projects

Select a target symmetric cryptographic algorithm and a list of design goals, e.g., from Section 11.3.1.

1. Design a reconfigurable hardware architecture for this algorithm, simulate it and implement it for a target device from any FPGA manufacturer.
2. Evaluate the resulting efficiency of your design according to the metrics of Section 11.3.2.
3. Compare it with the existing literature and put forward the weaknesses of your design.
4. What specific features of your FPGA did you exploit?
5. Then, think about hardware security from a general point of view. Select a physical attack that you want to prevent and add one or several countermeasure(s) to your design.
6. Comment on the efficiency versus security tradeoff.
7. Does your countermeasure add new physical weaknesses?

References

1. D. Agrawal, B. Archambeault, J. Rao, and P. Rohatgi. *The EM Side-Channel(s), in the proceedings of CHES 2002*, LNCS, vol. 2523, pp. 29–45, Redwood City, California, USA, August 2002.
2. R. Anderson and M. Kuhn. *Tamper Resistance – a Cautionary Note, in the proceedings of the USENIX 1996*, pp. 1–11, Oakland, USA, November 1996.
3. P. Barreto and V. Rijmen. *The KHAZAD Legacy-Level Block Cipher*, available from: http://www.cosic.esat.kuleuven.ac.be/nessie/
4. J. L. Beuchat. Modular multiplication for FPGA implementation of the IDEA block cipher, Research Report, num 2002-32, ENS Lyon, September 2002.
5. D. Boneh, R. DeMillo, and R. Lipton. *On the Importance of Checking Cryptographic Protocols for Faults, in the proceedings of Eurocrypt 1997*, LNCS, vol. 1233, pp. 37–51, Konstanz, Germany, May 1997.
6. L. Bossuet, G. Gogniat, and W. Burleston. *Dynamically Configurable Security for SRAM FPGA Bitsreams, in the proceedings of IPDPS 2004*, pp. 146–158, Los Alamitos, CA, USA, April 2004.
7. P. Bulens, K. Kallach, F.-X. Standaert, and J.-J. Quisquater. *FPGA Implementation of eSTREAM Phase-2 Focus Candidates with Hardware Profile, in the proceedings of SASC 2007*, Bochum, Germany, February 2007.
8. V. Maingot, J. B. Ferron, G. Canivet, and R. Leveugle. Fault attacks on SRAM-based FPGAs, USEIT Security Workshop, Toulouse, France, July 2007.
9. S. Chari, J. Rao, and P. Rohatgi. *Template Attacks, in the proceedings of CHES 2002*, LNCS, vol. 2523, pp. 13–28, Redwood City, CA, USA, August 2002.
10. K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, v. 34(2): 171–210, June 2002.
11. S. Drimer. FPGA design security bibliography webpage, http://www.cl.cam.ac.uk/∼sd410/fpgasec/
12. S. Drimer. *FPGA Bitstream Authentication: Why and How*, in the proceedings of ARC 2007, LNCS, vol. 4419, pp. 73–84, Rio de Janeiro, Brazil, March 2007.
13. I. Gonzales and F. J. Gomez-Arribas. Ciphering algorithms in microBlaze-based embedded systems. *IEE Proceedings, Computers and Digital Technologies*, 153(2): 87–92, March 2006.
14. T. Good and M. Benaissa. *AES on FPGA: From the Fastest to the Smallest, in the proceedings of CHES 2005*, LNCS, vol. 3659, pp. 427–440, Edinburgh, UK, September 2005.
15. K. Jarvinen, M. Tommiska, and J. Skytta. Comparative survey of high-performance cryptographic algorithm implementations on FPGAs. *IEE Proceedings*, 152(1): 3–12, October 2005.
16. T. Kean. *Cryptographic Rights Management of FPGA IP Cores, in the proceedings of FPGA 2002*, pp. 113–118, Monterey, CA, USA, February 2002.
17. P. Kocher, J. Jaffe, and B. Jun. *Differential Power Analysis*, in the proceedings of Crypto 1999, LNCS, vol. 1666, pp. 398–412, Santa-Barbara, USA, August 1999.

18. P. Lysaght, B. Blodget, J. Young, and B. Bridgford. *Enhanced Architectures, Design Methodologies And CAD Tools For Dynamic Reconfiguration of Xilinx FPGAs, in the proceedings of FPL 2006*, Madrid, Spain, September 2006.
19. T. G. Malkin, F.-X. Standaert, and M. Yung. *A Comparative Cost/Security Analysis of Fault Attack Countermeasures, in the proceedings of FDTC 2005*, LNCS, vol. 4236, pp. 159–172, Edinburgh, Scotland, September 2005.
20. E. Peeters, F.-X. Standaert, N. Donckers, and J.-J. Quisquater. *Improved Higher-Order Side-Channel Attacks With FPGA Experiments, in the proceedings of CHES 2005*, LNCS, vol. 3659, pp. 309–323, Edinburgh, Scotland, September 2005.
21. Jan M. Rabaey. Digital Integrated Circuits, Prentice Hall International, 1996.
22. F. Rodriguez, N. A. Saqib, A. D. Perez, and Ç. K. Koç. *Cryptographic Algorithms on Reconfigurable Hardware*, Springer, 2006.
23. G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat. *Design Strategies and Modified Descriptions to Optimize Cipher FPGA Implementations: Fast and Compact Results for DES and Triple-DES, in the proceedings of FPL 2003*, LNCS, vol. 2778, pp. 181–193, Lisbon, Portugal, September 2003.
24. G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat. *Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Applications, in the proceedings of ITCC 2004*, Las Vegas, USA, April 2004.
25. G. Rouvroy, F.-X. Standaert, F. Lefebvre, and J.-J. Quisquater. *Reconfigurable Hardware Solutions for the Digital Rights Management of Digital Cinema, in the proceedings of DRM 2004*, pp. 40–53, Washington DC, USA, October 2004.
26. L. Shang, A. Kaviani, and K. Bathala. *Dynamic Power Consumption in Virtex-2 FPGA Family, in the proceedings of FPGA 2002*, pp. 157–164, Monterey, California, USA, February 2002.
27. E. Simpson and P. Schaumont. *Offline Hardware/Software Authentication for Reconfigurable Platforms, in the proceedings of CHES 2006*, LNCS, vol. 4249, pp. 311–323, Yokohama, Japan, October 2006.
28. F.-X. Standaert, G. Rouvroy, J.-D. Legat, and J.-J. Quisquater. *Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs, in the proceedings of CHES 2003*, LNCS, vol. 2779, pp. 334–350, Cologne, Germany, September 2003.
29. F.-X. Standaert, S. B. Ors, and B. Preneel. *Power Analysis of an FPGA Implementation of Rijndael: Is Pipelining a DPA Countermeasure?, in the proceedings of CHES 2004*, LNCS, vol. 3156, pp. 30–44, Cambridge, MA, USA, August 2004.
30. F.-X. Standaert, E. Peeters, G. Rouvroy, and J.-J. Quisquater. *An Overview of Power Analysis Attacks Against Field Programmable Gate Arrays*, in the Proceedings of the IEEE, 94(2):383–394, February 2006.
31. F.-X. Standaert, E. Peeters, F. Mac, and J.-J. Quisquater. *Updates on the Security of FPGAs Against Power Analysis Attacks, in the proceedings of ARC 2006*, LNCS, vol. 3985, pp. 335–346, Delft, The Netherlands, March 2006.

32. F.-X. Standaert, G. Rouvroy, and J.-J. Quisquater. *FPGA Implementations of the DES and Triple-DES Masked Against Power Analysis Attacks, in the proceedings of FPL 2006*, Madrid, Spain, August 2006.
33. F.-X. Standaert, T. G. Malkin, and M. Yung. A formal practice-oriented model for the analysis of side-channel attacks, Cryptology ePrint Archive, Report 2006/139, 2006, available from http://eprint.iacr.org/2006/139
34. K. Tiri and I. Verbauwheder. *Synthesis of Secure FPGA Implementations, in the proceedings of the International Workshop on Logic and Synthesis (IWLS 2004)*, pp. 224–231, June 2004.
35. P. Tuyls, G. J. Schrijen, B. Skoric, J. van Geloven, N. Verhaegh, and R. Wolters. *Read-Proof Hardware from Protective Coatings, in the proceedings of CHES 2006*, LNCS, vol. 4249, pp. 369–383, Yokohama, Japan, October 2006.
36. T. Wollinger, J. Guarjardo, and C. Paar. Security on FPGAs: State of the art implementations and attacks. *ACM Transactions in Embedded Computing Systems*, 3(3):534–574, August 2004.
37. Xilinx. Virtex, Virtex-E, Virtex-II, Virte-II Pro, Virtex-4, Virtex-5 Field programmable gate arrays data sheets, http://www.xilinx.com
38. Xilinx. Xilinx FPGA identification friend of foe copy protection with 1-Wire SHA-1 secure memories, Application Note 3826, http://www.xilinx.com