Chapter 5
Hardware and VLSI Designs

Mario Kirschbaum and Thomas Plos

Abstract Efficient and secure hardware implementations have become a very popular topic during the last decades. In this chapter, we discuss the fundamental design approaches to successfully implement integrated circuits (ICs) as well as testing methods and optimization techniques to achieve an adequate solution for various application scenarios. A major topic handled in this chapter is security in the context of hardware implementations. We elaborate on the characteristics of modern CMOS circuits with regard to side-channel attacks and we discuss possible countermeasure approaches against such attacks. Furthermore, we describe a comprehensive practical example of combining cryptographic instruction set extensions with hardware countermeasures on a modern 32-bit processor platform. In the last section of this chapter, we argue about the assets and drawbacks of implementing test structures in digital circuits with regard to unintentionally opening security holes as well as about intentionally introducing malicious hardware structures, also called hardware Trojans.

Keywords VLSI design cycle · Design space · Advanced encryption standard (AES) · Secure hardware design · Side-channel analysis · Power-analysis attacks · Masking · Hiding · Dual-rail precharge · Instruction-set extensions · Design for test · Hardware Trojans

M. Kirschbaum (✉) · T. Plos
Institute for Applied Information Processing and Communications,
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
e-mail: Mario.Kirschbaum@iaik.tugraz.at

T. Plos
e-mail: Thomas.Plos@iaik.tugraz.at

5.1 Introduction and Motivation

During the last decades, integrated hardware circuits have become more popular and are an integral part of our daily life. Hardware circuits are not only found in personal computers and laptops, but also in cars, domestic appliances, and any kind of communication and multimedia device. Continuous migration to smaller process technologies has allowed us to dramatically increase the transistor count and consequently also the functionality of hardware circuits. In order to handle the increasing complexity of hardware circuits, a whole new branch of industry has been built up: Very large scale integration (VLSI) design.

Mainly two different approaches can be followed when designing complex hardware circuits: a general-purpose approach or a special-purpose approach. The general-purpose approaches based on hardware circuits like microprocessors that provide a fixed set of functionality. Customization of the microprocessor for a concerning application is done through program development which provides high flexibility. The special-purpose approach on the other hand, involves design of a dedicated hardware circuit for a more specific application. This hardware-based concept is less flexible and causes longer development times, but allows optimizing a design toward a certain goal, for example: low area, low power consumption, low energy consumption, or high throughput. Reducing the hardware overhead is desirable for cost-sensitive high-volume products that aim for minimum chip area. Achieving low power consumption or low energy consumption is important for passively-powered devices (e.g., RFID tags) and battery-operated devices, respectively. Especially when integrating complex and resource-intensive operations into a device, like for example cryptographic operations in a security-related application, fulfilling the design requirements is often only achievable if dedicated hardware circuits are used.

In the following sections, we discuss some principles of hardware and VLSI design. We start with describing the fundamental VLSI design cycle which is the basis of every integrated circuit (IC). Hardware designers may choose between different design perspectives at various abstraction levels encountered during the design cycle. Starting with the system specification, a designer defines the submodules, uses hardware description languages (HDLs), applies tools for standard-cell mapping, and finally comes to the geometric layout of the design.

During the design cycle, repeated testing and simulation of the design at different abstraction levels is inevitable in order to obtain a flawless and well functioning circuit. In case errors occur, only a small step has to be made back in the design cycle instead of returning to square one. Another important topic we will discuss is the extensive design space which is at the designer's disposal during the whole design cycle. A designer has almost indefinite possibilities to reach a design goal, which is, e.g., high performance, low area, or low energy consumption. By means of several practical examples, we illustrate the impact of decisions made during the design cycle on the outcome of a hardware design.

In the next part, we concentrate on *security* in hardware and VLSI design, which is an important topic. The use of security-related devices is steadily increasing, e.g., web servers protected with SSL, encrypted hard-disc drives, wireless car keys, or radio-frequency identification (RFID) tags, to name but a few well-known examples. Mathematically secure cryptographic algorithms become highly vulnerable to various types of attacks when implemented in hardware due to the strong relation between the data processed within a device and the power consumption of conventional complementary metal-oxide semiconductor (CMOS) circuits. In order to address this issue, researchers began to develop countermeasures to protect sensitive hardware devices. We have a detailed look at the fundamental characteristics of CMOS circuits that enable side-channel attacks in the first place and we elaborate on possible approaches for implementing countermeasures against such attacks. Similar to the importance of repeatedly simulating a design during the design cycle to obtain a reliably-working circuit, we point out the possibility of verifying the effectiveness of countermeasures by means of simulations.

Combining the topics of efficient hardware implementations, cryptographic algorithms, and security, we contrast different approaches for implementing cryptographic algorithms on modern embedded devices. As we will see, pure software as well as pure hardware solutions have both drawbacks. A more efficient solution in case of embedded systems is the usage of instruction-set extensions (ISEs). Furthermore, we discuss the implementation of countermeasures against side-channel attacks in the presence of ISEs and we elaborate on a practical example of a modern 32-bit processor platform.

The testability of a device during the design cycle as well as after manufacturing of an IC is very important to prevent distribution of malfunctioning parts. Unfortunately, the integration of test structures may counteract the efforts to protect a device from various attacks, since test structures could be used by an adversary to break security-enabled devices. We describe different testing approaches and discuss their relevance and possible impact on secure implementations. Finally, we discuss the topic of hardware Trojans. Contrary to test structures implemented in ICs that may unintentionally cause a vulnerability of the device, hardware Trojans are malicious structures intentionally implemented by an adversary with the goal, for example, to possibly bypass one or more security features of an IC.

5.2 VLSI Design Cycle

Today's VLSI designers are faced with two challenges, increasing circuit complexity and shorter design cycles. Following Moore's Law, the transistor count of hardware circuits doubles about every 18 months. This prediction has been formulated more than 40 years ago and is still adhered to by the semiconductor industry by migrating towards smaller and smaller process technologies. Most recent microprocessors, for example, have already reached a transistor count of one billion and more. Circuit complexity grows faster than the productivity of designers and the increase in efficiency of electronic design automation (EDA) tools. This has opened a so-called "design gap" over the years. In order to close this gap, design of VLSI circuits has

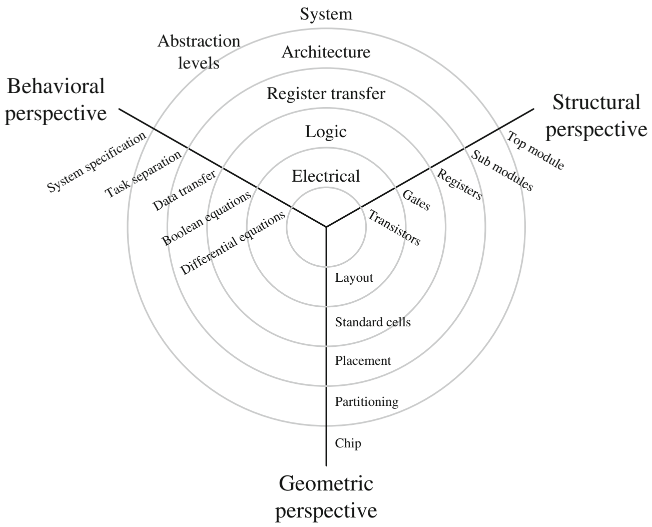

Fig. 5.1 Y-diagram according to Gajski and Kuhn showing the different design perspectives and abstraction levels of hardware circuits

been brought to a higher abstraction level. Designing a circuit at a higher abstraction level also addresses the requirement of shorter design times to improve cost effectiveness.

A good overview of the different abstraction levels and design perspectives of hardware circuits is provided by the Y-diagram illustrated in Fig. 5.1. The Y-diagram has been introduced by Gajski and Kuhn [6] and has its name from the three axes that are arranged in a y-shape. Each axis relates to a different design perspective. The three design perspectives are behavioural perspective, structural perspective, and geometric perspective. Behavioural perspective focuses on the functionality of a circuit, whereas structural perspective describes the interconnection of different blocks within it. Geometric perspective deals with the arrangement of the components, including the final layout of a circuit. Concentric circles indicate the various abstraction levels, which are system level, architectural level, register-transfer level, logic level, and electrical level. Starting from highest abstraction level at the outermost circle, the various development steps of a hardware circuit are passed through when moving toward the center of the diagram, marking the final outcome of the design (i.e., layout of the circuit).

When moving toward the center of the diagram to reach the design goal, the level of detail increases continuously. Different perspectives can be used for entering lower abstraction levels and changing between perspectives is possible as well. Behavioral perspective is the most-suitable domain for describing hardware designs with high complexity. Consequently, behavioral perspective is used for starting the design process. The first step is creating a software model that implements the specification of the system and that allows exploring different algorithm variants. This first software model also eases communication among design teams and enables concurrent development of hardware and software components (important to shorten overall development time). Next step is finding an appropriate architecture that is reflected by a cycle-accurate high-level model. When the architecture is fixed, HDLs like VHDL and Verilog are deployed to transfer the high-level model into a register-transfer level representation. The combined use of HDLs and EDA tools for circuit synthesis allows an automated transformation from behavioral perspective to structural perspective. The outcome of this step is a netlist that contains a circuit representation with logic gates, flip flops, and the appropriate wire connections.

The following steps after netlist creation relate to the so-called back-end design where the structural perspective is left and the geometric domain is entered. Automated tools are again applied to deduce a standard-cell representation and the layout of the design. During back-end design, various verification techniques are utilized to ensure proper operation and manufacturability of the circuit. Verification techniques comprise for example, design-rule checks, electrical-rule checks, layout-versus-schematic checks, timing verification, and simulation of power consumption. With the layout of the circuit, the final design step (tape out) is reached and data can be sent to a semiconductor manufacturer.

Following this top-down approach gives a good understanding of the involved steps of state-of-the-art VLSI design. Implementing a circuit within behavioral perspective through HDLs and deploying automated tools for further processing eases not only the handling of circuit complexity, but also brings also more flexibility. A circuit in HDL representation can be easily mapped to different process technologies and targets by using circuit-synthesis tools. This significantly shortens the time required for migrating a design to a new process technology and allows also first-level tests on field-programmable gate array (FPGA) prototypes.

Continuously testing the functionality of a design within all abstraction levels is an important aspect of modern VLSI design. Required test data is typically derived from the high-level model and repeatedly used for tests on lower abstraction levels. When a test fails, designers can immediately step back and fix the problem. This allows detection of issues as early as possible, following the first-time-right concept to launch products on time.

When building hardware circuits that contain security-relevant components, functional tests alone are no longer enough. Additional considerations have to be taken into account like evaluating the resistance of the implementation against side-channel analysis (SCA) and fault analysis. Such evaluation tests are mainly conducted after chip production on first prototype samples, but also during design phase. Power-simulation results of the circuit can be used to deduce first information about side-channel resistance of a design. Other examples are side channel and fault attacks on FPGA prototypes that contain a synthesized version of the design.

5.3 Design Space of Hardware Circuits

Hardware circuits can be designed toward different optimization goals, depending on the targeted application. Typical optimization goals are high throughput, low area, low power consumption, and low energy consumption. Optimization can be conducted on different abstraction levels. However, the higher the abstraction level, the larger is the impact of the optimization techniques and the lower the required effort. Optimizing a design at system level or at architectural level is therefore more promising than optimizing it for example on logical level. Various metrics are used to quantify the effectiveness and the influence of a certain optimization measure. Widely-used metrics includes chip area, throughput, execution time, maximum clock frequency, latency, and average power consumption.

Optimization at system level typically involves finding more suitable protocols or looking for alternative algorithms that lead to the same result but provide advantageous behavior in terms of computation time or resource usage. A good example is the representation of the substitution box (S-box) used in the AES. The S-box is a non-linear operation that is applied on a single byte of data. Hence, the result of the S-box operation can be precomputed for all possible 2^8 input values and stored in a look-up table. This will result in an area requirement of more than $1,000$ GEs when implementing the look-up table with standard cells. However, the S-box operation can also be realized by calculating the multiplicative inverse in the finite field $GF(2^8)$ followed by an affine transformation (see [15] for more details). Using combinatorial logic to calculate the S-box operation in that way, leads to an area requirement of 300 GEs. This is less than a third of the value required by the look-up table approach. Achieving such an area saving through optimization at lower abstraction levels is hardly possible.

Architecture is another abstraction level that has significant potential to optimize a design toward a certain direction. Well-known optimization techniques at architectural level are functional decomposition, pipelining, and parallel computation [8]. Functional decomposition aims at breaking a complex function into smaller subfunctions that can be computed sequentially. This method is most effective when the subfunctions compute similar operations that allow reusing of a single hardware unit that decreases the overall chip area. Execution time remains roughly the same, since the shorter critical path allows a higher maximum clock frequency, which compensates for the increased number of required clock cycles.

Pipelining is another effective optimization method at architectural level. The data path of a function is cut into smaller parts (ideally of equal length) by inserting storage elements called pipeline registers. This shortens the critical path and leads to a higher maximum clock frequency. For computing the result of one data item, as many clock cycles are required as there are pipeline stages. However, once the whole pipeline is filled, the result of a data item is computed with every clock cycle. It is important to note that this works only if there are no recursive data dependencies, since they would prevent the pipeline from getting filled. Pipelining is very efficient because a marginal increase of chip area that is introduced by adding pipeline registers, results in a significant computational speed-up.

Computing operations in parallel is the opposite of functional decomposition. Instead of reusing components to reduce chip area, additional hardware modules are introduced to lower computation time. Trading chip area for speed is some kind of brute-force approach and is used if other measures like pipelining are not applicable (e.g., if low latency is required). In contrast to pipelining, the critical path of a design is not shortened and therefore increasing the clock frequency is not possible. Chip-area requirements increase significantly and relate to the degree of parallelism.

An overview of the impact of all three optimization techniques within the design space is given in Fig. 5.2. Functional decomposition and pipelining are efficient approaches to decrease chip area and execution time of a design, respectively. Both techniques significantly lower the area-time product. Parallel execution of operations increases chip area to lower execution time, by keeping the area-time product roughly constant.

The following examples illustrate the effects on hardware implementations of the advanced encryption standard (AES) when focusing on different design goals and implementing different optimization techniques. AES is a symmetric block cipher and has been standardized by the National Institute of Standards and Technology (NIST) in 2000 [15]. Let us first have a look at the low-power AES implementation of Feldhofer et al. [4]. The design goals of this AES implementation have been low area and low power in order to apply AES in highly resource-limited devices like RFID tags. The AES module supports encryption and decryption including the key schedule and is based on an 8-bit architecture. In order to reduce the area to a minimum the design contains only one S-Box instance (combinational, one pipeline stage) and one multiplier for the MixColumns operation. The usage of only one S-Box instance corresponds to functional decomposition, i.e., one S-Box instance is used several times during one cryptographic computation. The pipeline stage within the S-Box implementation helps to shorten the critical path of the design. One encryption/decryption can be done in roughly 1, 000 cycles. Strictly following low area and low-power guidelines, the developers produced the AES module in a

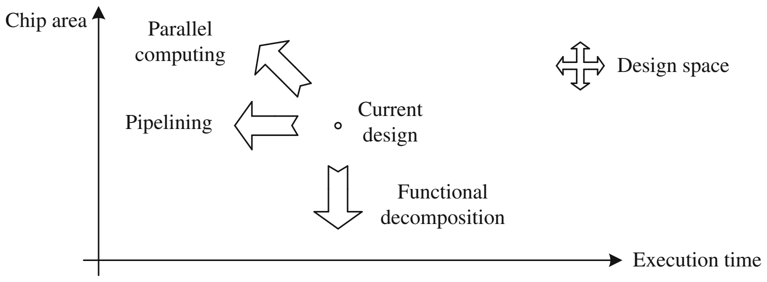

Fig. 5.2 Impact of *functional decomposition*, *pipelining*, and *parallel computation* on *design space*

0.35 μm CMOS technology and were able to achieve a very low area requirement of 3, 400 GEs and an extremely low power consumption of 3.0 μA when operating the AES module at 100 kHz and a supply voltage of 1.5 V. In this configuration, the module achieves a throughput of approximately 12.5 kbps.

Contrary to the low-area implementation of Feldhofer et al., Mangard et al. [12] proposed a high-performance hardware implementation of the AES based on a 32-bit architecture. The AES module follows the parallel computation approach and contains 16 S-Box instances and 16 multipliers implementing the MixColumns operation. The AES implementation has an area requirement of 16 kGEs, needs 34 cycles per encryption/decryption, and reaches a maximum clock frequency of 64 MHz and a throughput of 241 Mbps produced in a 0.6 μm CMOS technology. There also exist some extreme-performance implementations of AES, deploying a 128-bit architecture and highly optimized implementations of the AES operations, resulting in larger implementations (beyond 20 kGEs) and significantly higher throughput ($\geq$1 Gbps).

These examples show that a designer has almost indefinite possibilities to exploit the design space in many different directions. Often a designer decides to go in more than one direction at once: e.g., low area and low power, high throughput, and low area. In the end, a designer is mostly forced to accept compromises between throughput, area, and power consumption in order to achieve an adequate hardware solution suitable for the particular application.

5.4 Secure Hardware Design

The use of security-related devices has been steadily increasing during the last few years. Besides meeting appropriate design goals in terms of throughput, chip area, and power consumption, security goals started to play a major role in hardware design. Various attacks on hardware circuits in the past have pushed the emergence of a completely new research field.

Additionally to the intended output, e.g., the result of a cryptographic computation, each physical device also emits various other information, the so-called side-channel information. This information is permanently present, before, during, and after a computation. A very obvious side channel is timing. It is quite easy to accurately measure the time required for executing a cryptographic operation on a device. Kocher has first shown the vulnerability of asymmetric cryptographic algorithms to timing attacks [9]. Preventing timing attacks lies manly at the designer's hands. For example, in most cases, it is relatively easy to avoid conditional branches, and hence, to avoid a data-dependent timing behavior of a cryptographic implementation.

The last decade has shown that avoiding data-dependent information in the power consumption of a device is not so easy. After the first publication on SCA attacks that exploit the power consumption of cryptographic devices by Kocher et al. [10], the security of hardware designs against power analysis (PA) attacks became a major research topic in the fields of cryptography and hardware implementations. As it turned out that almost any cryptographic device implemented in CMOS technology is highly vulnerable to PA attacks, designers of algorithms as well as hardware developers began to think about possible solutions to overcome the inherent relationship between the processed data within a CMOS circuit and its total instantaneous power consumption.

5.4.1 Power Consumption of CMOS Gates

The total power consumption of CMOS gates is the sum of the *static power consumption* and the *dynamic power consumption*. The static power consumption is caused by a small leakage current that is flowing through the metal-oxide semiconductor (MOS) transistors that are turned off. An actual example is given in [27]: the leakage current of a MOS transistor in a 100 nm process is typically in the nA range. In most applications, the static power consumption of CMOS circuits is neglected, except for low-power applications. In such applications, special low-leakage process technologies come into play which are able to significantly reduce the static power consumption. From a security perspective, the static power consumption of CMOS circuits can be neglected, as the leakage current only shows an extremely low data dependency. Dynamic power consumption, on the contrary, is significantly higher than static power consumption, and even more important, it shows a strong dependency to the data processed by the CMOS circuit.

In the following, the fundamental characteristics of CMOS circuits, which enable the execution of power-analysis attacks in the first place, are described by means of a conventional CMOS inverter. The schematic of a CMOS inverter is depicted in Fig. 5.3 (left), it basically consists of a pMOS transistor p and an nMOS transistor n. The output line q of the inverter is naturally afflicted with a vast number of parasitic capacitances. In a simplified model, we can assume two significant-pooled parasitic capacitances (indicated as C_{L1} and C_{L2} in Fig. 5.3). Depending on the state transition of the CMOS inverter one of the capacitances is charged. The events in case input

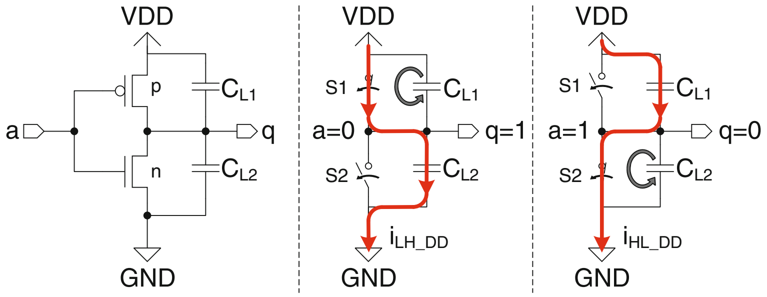

Fig. 5.3 Depiction of the power consumption of a CMOS inverter: the schematic of the inverter (*left plot*), the equivalent circuit in case input $a : 1 \rightarrow 0$ (*middle plot*), the equivalent circuit in case input $a : 0 \rightarrow 1$ (*right plot*)

a switches from 1 to 0 (i.e., $q : 0 \rightarrow 1$) are illustrated in the equivalent circuit in Fig. 5.3 (middle). The pMOS and nMOS transistors are represented by switches $S1$ (closed) and $S2$ (opened), respectively. Assuming that C_{L1} is charged from the previous state and C_{L2} is discharged, the following charging processes occur: C_{L1} is discharged internally via $S1$ and C_{L2} is charged via i_{LH_DD}, i.e., the CMOS inverter consumes power to charge C_{L2}. In case input a switches from 0 to 1 (i.e., $q : 1 \rightarrow 0$), very similar charging processes occur in the circuit (Fig. 5.3, right): $S1$ is opened, $S2$ is closed, C_{L2} is discharged internally via $S2$, and C_{L1} is charged via i_{HL_DD}. The CMOS inverter consumes power to charge C_{L1}.

We can summarize the events happening in the CMOS inverter in the following way: if the input of the inverter does not change (i.e., $a : 0 \rightarrow 0$ or $a : 1 \rightarrow 1$), the nMOS and pMOS transistors keep their state, and none of the output capacitances C_{L1}, C_{L2} needs to be charged. Neglecting the static power consumption, we can state that the power consumption of a CMOS inverter is *zero* in case the input signal remains in its state. On the other hand, if the input of the inverter changes its state (i.e., $a : 0 \rightarrow 1$ or $a : 1 \rightarrow 0$) the nMOS and pMOS transistors change their conductivity and the output capacitances C_{L1}, C_{L2} are charged/discharged accordingly. We see a change of the input value causes a significant amount of power consumption. Furthermore, we can also imagine the following: assuming we know the initial state of the inverter and we record the power consumption of our small CMOS circuit, we are able to determine the actual state of the circuit at any time by simply considering the power spikes we see on our record. That is exactly the reason why power-analysis attacks pose a serious threat to CMOS circuits. An attacker can quite easily figure out what is happening in a circuit by analysing the power consumption.

As the power consumption is closely related to the electromagnetic (EM) emanation of a device, EM analysis attacks can also reveal very small data dependencies within a device. More specifically, EM-analysis attacks and usually even more powerful than power-analysis attacks, because with the appropriate equipment, the measurement of the EM emanation of a device can be limited to a very small portion of the circuitry. This way, the signal-to-noise ratio (SNR) of the measurement can be significantly increased.

5.4.2 Countermeasures Against Power-Analysis Attacks

The following section gives a broad outline of ideas that have been developed during the last few decades to impede PA attacks. We introduce the basic approaches and indicate the main assets and drawbacks. The underlying concept of countermeasures against PA attacks is to break the dependency between the processed intermediate data within a device and the device's instantaneous power consumption. Basically, there exist two approaches: masking countermeasures and hiding countermeasures. Figure 5.4 depicts the points where the two approaches apply. Both approaches can be utilized at the architecture level (implemented in software and/or hardware) as

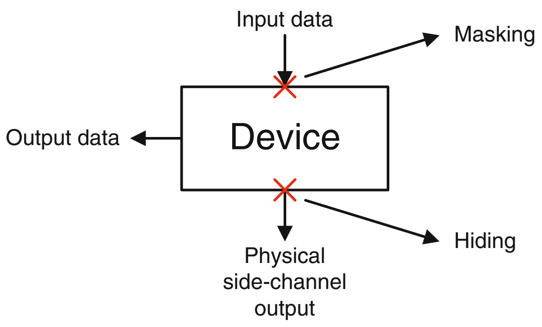

Fig. 5.4 Depiction of the two basic countermeasure approaches: masking and hiding

well as at the cell level (purely implemented in hardware). Usually, a combination of both approaches is worthwhile to achieve an adequate level of security.

Masking countermeasures conceal the processed data within the device by a random mask, i.e., the input data is altered before any operations are executed. Hence, the device's power consumption only depends on the masked data. After performing the critical operations within the device, the mask has to be removed again. Some additional effort is associated when implementing a masking countermeasure, in many cases the algorithm's operations need to be adapted in order to process the masked data correctly, e.g., when masking an AES S-Box operation [3].

Special logic styles belong to the strongest methods to prevent PA attacks. The secure logic-style front is highly competitive. Many approaches have been presented during the last decade, although for most one or more flaws have been discovered by the research community. Let us have a closer look at one example for such a special logic style implementing a masking technique: the masked dual-rail precharge logic (MDPL), proposed by Popp et al. [16], Popp and Mangard [17]. The MDPL style is based on the dual-rail precharge (DRP) principle which prevents the occurrence of glitches by representing each signal with complementary electrical wires in the circuit. It is well known that glitches, also called *dynamic hazards*, do occur in CMOS circuits and that they have a significant effect on the power consumption [19]. Considering their effect on the power consumption, it appears obvious that glitches also play a major role in the context of countermeasures against PA attacks. It has been shown several times that glitches may have a negative influence on the effectiveness of countermeasures [13, 21]. Preventing glitches is achieved by strictly applying only monotonic logic functions and by introducing a precharge phase[1] and an evaluation phase[2] in each clock cycle. In a DRP circuit, only one of the two complementary wires of each signal is HIGH, depending on the value of the signal, the other wire remains LOW. Additionally, each cell within an MDPL circuit unexceptionally processes masked signals, with the result that the power consumption

[1] In the precharge phase, every signal (both complementary wires) within a digital circuit is charged to the precharge value, which is in most cases logic '0'.

[2] Similar to a standard clock cycle in a conventional CMOS circuit, the combinational blocks start to evaluate according to their input signals.

only depends on masked values. A not yet eliminated flaw in the MDPL style is that the mask value can be discovered due to significant differences in the power consumption, as the mask signal is connected to every MDPL cell in the circuit and hence it has to overcome a major amount of parasitic capacitances [20, 25]. The general drawbacks of special logic styles may be manifold: a significant overhead in terms of area requirement and power consumption, a decrease of performance, and a considerable effort for implementing the logic style in the first place, which is especially the case for logic styles that are based on full-custom cells.

Hiding countermeasures directly alter the side-channel characteristics of the device with the result that the correlation between the processed data within the device and the device's power consumption is weakened or even completely dissolved. This way, an attacker is not able to draw any conclusions from the power consumption about the intermediate data processed in the device. There are basically two approaches to implement hiding countermeasures: randomizing the power consumption of a device (also called *hiding in time*) and equalizing the power consumption of a device (also called *hiding in amplitude*). Unfortunately, both approaches are almost impossible to be *perfectly* implemented in practice.

An example for hiding in time is the random insertion of additional operations during the execution of a cryptographic algorithm. The additional operations do not contribute anything to the actual cryptographic computation and therefore they are called *dummy operations*. The random insertion of dummy operations increases the runtime of the whole computation with the result that the power consumption of the critical operations on actual data is randomized in time. This approach simply adds noise to power measurements and thus complicates a side-channel attack. The random insertion of dummy operations has to be implemented with great care: the dummy operations must not be distinguishable from the actual operations, otherwise the execution times of dummy operations can be detected and filtered. Furthermore, once the countermeasure is activated, the number of dummy operations that are executed has to remain constant for every cryptographic computation. Otherwise the implementation is vulnerable to timing attacks. As the runtime of the computation correlates with the number of dummy operations inserted, another drawback is that the countermeasure has to be adapted to every cryptographic algorithm for which it is implemented.

A very simple example for a countermeasure following the approach of hiding in amplitude is the application of noise generators within a device. The obvious drawbacks of noise generators are the requirement of area and power, without contributing anything to the actual functionality of the circuit. A further example for hiding in amplitude is a heavily parallelized design of an algorithm. However, this approach strongly depends on the implemented algorithm, as data dependencies limit the possible degree of parallelization.

More-advanced techniques are based for example on the previously mentioned DRP logic style. Pure DRP styles, which do not implement a masking technique, try to keep the power consumption constant and thus independent of the processed data. This is achieved by balancing each pair of complementary wires, i.e., the designer tries to adjust the complementary wires in a way that their electrical characteristics (resistance, inductance, and capacitance) perfectly match. Ideally, a circuit containing perfectly balanced wires would consume a constant amount of power, and hence, it would be secure against PA attacks. The irrefutable flaw of this approach is that an exact balancing of wires is almost impossible to achieve in practice. Even if the most powerful EDA tools are used, the smallest variations in the chip-fabrication process would cause differences in the electrical characteristics of the complementary wires once more.

5.4.3 Verification of Countermeasures by Means of Simulations

As mentioned in Sect. 5.2, simulations play a major role during the design phase to verify the functionality of the IC. When implementing countermeasures, it is also highly desirable to be able to verify the efficiency of the implemented protection techniques. Various simulation techniques can be used to perform a detailed investigation of the implemented countermeasures without the need of actually fabricating an IC. The following section describes to what extent simulations at different levels can be used to estimate the impact of countermeasures.

One of the main advantages of simulations is the possibility to detect errors in a design before a chip goes into production. This is also the case for detecting flaws in countermeasures implemented in a secure hardware design, before developing a costly prototype chip. Simulations also offer the possibility to simply narrow down the simulated parts of a digital circuit and hence to easily detect and improve faulty submodules or vulnerable parts of an implemented countermeasure. For investigating countermeasures, two simulation levels are interesting: transistor-level simulations and logic-level simulations.

Transistor-level simulations based on SPICE [18] models of transistors represent a highly accurate yet very time consuming way to verify the correct functionality of digital circuits and countermeasures. SPICE simulations may include detailed parasitic information about each element in a circuit and about each wiring in a placed and routed chip design. This results in power-estimation results that are highly comparable with power measurements on an actual chip. Hence, transistor-level power simulations are very suitable to perform power-analysis attacks and to investigate the effectiveness of countermeasures. The main drawback of transistor-level simulations is the complexity and the associated expenditure of time due to solving countless algebraic equations based on nonlinear transistor models. If a designer does not have a powerful computer cluster at hand, the transistor-level simulation of a medium-sized design consisting of approximately 2 million transistors may easily take several hours for a few-hundred clock cycles. Considering that hundreds of power simulations are potentially required to perform a meaningful PA attack, transistor-level simulations become a rather impractical.

Logic-level simulations (also called gate-level simulations) in their simplest form have the advantage of operating at a significantly higher level (i.e., not including any low-level circuit information) compared to transistor-level simulations. This implies a significant speed up of performed simulations, but also a decrease in simulation accuracy. Furthermore, a conventional logic-level simulation is not able to provide something similar to a power consumption trace, it is merely possible to obtain logic-level transitions of each signal within a digital circuit. As described in Sect. 5.4.1, the state transition of CMOS gates is directly related to the dynamic power consumption. Hence, it is possible to derive a simulated power trace from the logic-level transitions obtained from the simulations. A common technique is called *toggle counting* or *transition counting*. At each point in the simulation time where a signal transition ($0 \rightarrow 1$ or $1 \rightarrow 0$) occurs, the power-consumption value for this specific point in time is increased by 1. Constant signals ($0 \rightarrow 0$ or $1 \rightarrow 1$) do not contribute anything to the power consumption. This way a designer is able to obtain power consumption traces in a fraction of the time needed to perform transistor level simulations.

There are some limitations in case of power traces derived from basic logic-level simulations. First, there is no timing information at all included in the simulations: some simulators work with unit delay (i.e., all logic gates have the same constant propagation delay) or zero delay (i.e., all transitions occurring in a specific time are summed up to one point in time, usually the clock event). Second, all signal transitions consume the same amount of power, which is highly unrealistic compared to an actual digital circuit. The accuracy of power-consumption traces derived from logic-level simulations can be substantially increased if back-annotated delay information is included in the simulations. This approach has minor effect on the performance of logic-level simulations but greatly increases the accuracy of signal-delay information. A second measure to increase the accuracy of toggle-count power traces is to randomly weight the signal transitions or to include parasitic information when processing the signal transitions. The latter approach would result in a time-consuming preprocessing step to build an appropriate transition-weight database.

Although we have seen that various simulation techniques may be used to verify the efficiency of implemented countermeasures, unforeseen effects may cover actual side-channel leakages during simulation.

5.5 Instruction-Set Extensions

Efficiently implementing cryptographic algorithms on embedded devices is highly challenging due to the limited resources (energy, clock frequency, and memory). A widely deployed processor for embedded devices is for example the LEON CPU core [5], which is a SPARC V8-compliant processor. The LEON core has a 32-bit architecture and follows the Reduced Instruction Set Computing (RISC) concept. When implementing a cryptographic algorithm on such an embedded system, a designer has mainly two options: selecting a software approach or a choosing a hardware approach. The software approach uses only the existing instructions of the processor and requires no additional hardware. This concept provides maximum flexibility, but is costly in terms of code size and allows achieving only a moderate computation speed. A hardware solution on the other hand requires the integration

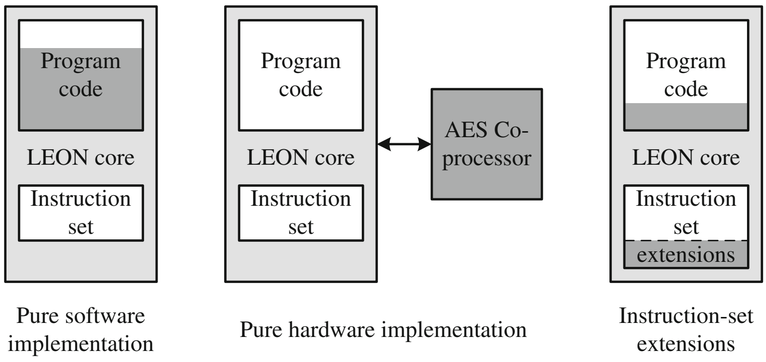

Fig. 5.5 Design alternatives for implementing AES on a LEON core. *Dark-gray* colored areas contribute to the AES implementation

of a dedicated coprocessor that is optimized for a special algorithm. Relying on an optimized hardware module allows very short execution times, which comes at cost of additional chip area and loss of flexibility. Another aspect that has to be considered is the communication overhead between embedded processor and coprocessor. As reported in the work of Hodjat and Verbauwhede [7], much more time is typically spent for the communication between processor and coprocessor, than for the actual computation of the algorithm within the coprocessor. This overhead dramatically lowers the performance gain of a coprocessor approach.

ISEs are techniques that combine the advantages from both pure software implementations and pure hardware implementations. ISEs provide the flexibility of a software solution together with the high computation speed of a dedicated hardware circuit. Moreover, there is no communication overhead between processor and coprocessor. A schematic overview of the three different design approaches is depicted in Fig. 5.5. ISEs provide a processor with additional instructions that are optimized for a certain purpose, like the execution of a cryptographic algorithm. The additional instructions require extra hardware circuits and can be used in a program as any other instruction. Hardware costs of the ISEs are much lower than those of a corresponding coprocessor.

A concept for ISEs on a LEON core has been presented by Tillich and Großschädl [22]. These ISEs aim for improving the computation speed of the AES algorithm on the embedded processor. The proposed ISEs allow computing AES within 196 clock cycles on the LEON core using only 896 bytes of code. The additional hardware costs introduced by the ISEs are estimated by the authors at 3 kGEs. For comparison, a pure software implementation of the AES algorithm on the LEON core takes 1,637 clock cycles and requires 2168 bytes of code. A pure hardware implementation on the other hand, as presented for example by Mangard et al. [12] (cp. Sect. 5.3), requires only 34 clock cycles for computing AES, but leads to additional hardware costs of 16 kGEs. Table 5.1 summarizes the performance numbers of the three design approaches and clarifies that ISEs are a highly efficient approach

Table 5.1 Overview of the different design approaches with corresponding performance numbers

Design approach	Code size [Bytes]	Execution time [Cycles]	Hardware costs [kGEs]
Pure software implementation	2168	1637	–
Pure hardware implementation	–	34	16
Instruction-set extensions	896	196	3

that provides a good tradeoff between computation speed and resource usage (in terms of code size and hardware overhead).

Also with regard to implementing hardware countermeasures (i.e., secure logic styles) against side-channel attacks, cryptographic ISEs have a significant advantage over dedicated coprocessors. In the case of cryptographic coprocessors, critical data is running countless times from one submodule to another and vice versa. For example, an AES coprocessor the input data is first XORed with the secret key. As the key represents the critical data in an AES computation, the following operations process critical data. In the next step, the critical data runs through the S-Box module and a probably directly-integrated ShiftRows module, followed by the MixColumns module back to the XOR operation with the next RoundKey. We see, in order to secure an AES coprocessor we need to implement countermeasures in many submodules, as all submodules directly process critical data. Securing a cryptographic coprocessor by means of hardware countermeasures usually results in implementing the whole coprocessor in a costly secure logic style, which entails a significant increase in terms of area requirements and power consumption.

In case of cryptographic ISEs implemented on a processor platform, operations that are actually transforming critical data are confined to the functional units of the processor. This circumstance enables us to implement only the functional units in a costly secure logic style and to implement a much cheaper countermeasure to the rest of the chip. The next section introduces a concept for developing a secure processor with ISEs and hardware countermeasures.

5.6 A 32-Bit Processor with ISEs and SCA Countermeasures

In the following we investigate a practical example of how ISEs can be combined with countermeasures against side-channel attacks on a modern processor platform. We discuss the main features of a comprehensive concept proposed by Tillich and Großschädl [23], Tillich et al. [24] for implementing AES ISEs and hardware countermeasures on a 32-bit SPARC V8-compliant processor.

In this concept the majority of the 32-bit processor remains unmodified. All critical operations are executed within a single hardware module which acts like a conventional function unit, the so-called *secure zone*. Only the secure zone is implemented in a secure logic style and contains some additional hardware blocks, the rest of the processor remains untouched and is implemented in standard CMOS logic. A protected functional unit within the secure zone provides a set of custom instructions that can be used for a flexible implementation of different cryptographic algorithms. In fact, all operations that may potentially become a target of side-channel attacks have to be unconditionally executed by the protected functional unit. Hence, a software developer still has to proceed with great care during the process of software development in order to avoid unintentionally implemented security leaks.

Within the secure zone, all operations are protected by a secure logic style, outside of the secure zone all data is strictly masked. Function operands entering the secure zone are unmasked, critical operations are performed on the unmasked data, and before leaving the secure zone the data is masked again with a freshly generated mask. This way it is ensured that transformations on critical data as well as the masks do not leak any side-channel information because of the secure logic style, and the masked data running outside of the secure zone may not leak any useful side-channel information because of the masking technique and the anonymity of the masks.

All mask-handling modules are also part of the secure zone, i.e., they are protected by the secure logic style, and the masks themselves must not leave the secure zone in their original form. The secure zone contains a mask generator and a mask storage for generating, storing, and retrieving masks. The retrieval of masks corresponding to input operands and the storage of a mask corresponding to a result can be linked to the addresses of the operands and the result, respectively.

We now want to take a look at the implementation costs of the practical example proposed by Tillich et al. [24] based on a SPARC V8-compliant LEON3 processor [1]. A full version of the secure zone has an area requirement of approximately 22 kGEs. In the following, we go through some theoretic numerical examples, calculating the total area overhead when implementing the secure zone in different secure logic styles.

We assume that a typical LEON3 processor implementation containing a debug support unit, RAMs, and caches requires approximately 580 kGEs. In total, a LEON3 processor equipped with a secure zone implemented in unprotected CMOS logic requires 602 kGEs. A pure DRP logic style like DWDDL increases the area requirement approximately by a factor of about 12 [31]. In our example the secure zone implemented in DWDDL would require approximately 264 kGEs, the whole LEON3 processor would thus result in 844 kGEs. The total area overhead would increase by a factor of only 1.4 compared to the LEON3 implementation in unprotected CMOS logic. In case of a coprocessor that would have to be completely implemented in DWDDL we would encounter the full area overhead factor of 12. In case of iMDPL [16], which causes an area overhead of a factor of 18, the overall area for our LEON3 processor would be 976 kGEs. This would result in an area overhead factor of approximately 1.7.

These numerical examples illustrate the following: although secure logic styles may result in a significant drawback in terms of area requirements as well as power consumption, they may be implemented much more economically if secure logic styles are combined with advanced concepts like the secure-zone approach.

5.7 Testability and Security

Testing is an important activity not only during development of a hardware circuit but also after its manufacturing. Typically, not all microchips that have been manufactured are working properly. This has various reasons, for example, varying process parameters during production or imperfections of material and masks. The yield, which is the ratio between the number of working chips and the overall number of manufactured chips, should be as high as possible to maximize the profit. In order to separate faulty chips from working chips, tests have to be applied.

Releasing a faulty chip causes tremendous costs. Imagine the following simple example: A company manufactures 100,000 chips, and sells them at the price of $1 per chip. We assume that one percent of the chips (i.e., 1,000 chips) are faulty. When the faulty chips are immediately detected after production through tests before they get sold, costs of $1,000 will arise. When the faulty chips get detected after they have been sold and soldered on a board, costs will already result in $50,000 if repairing a malfunctioning board costs $50. Even worse, when the failing parts get detected after integration into a whole system, costs will boost to $1,000,000 when repairing a non-working system costs $1,000. This simple example clearly emphasizes the need of detecting faulty parts as early as possible after production.

In order to get confidence about proper operation of a microchip after production, reliable tests are necessary. For realizing such reliable tests, the underlying test concepts that are used have to be planned and included at the design time of a hardware circuit. This so-called "design-for-test" approach integrates additional test structures to a circuit to allow fast and comprehensive analysis of a chip after production. The more internal details of a chip can be accessed, the more comprehensive tests can be conducted, lowering the chance that malfunctioning parts remain undetected.

A powerful and widely-used test concept uses scan chains that provide access to the values internally stored in the flip flops of a hardware circuit. For cryptographic devices, giving access to internal values can be problematic. As shown in the work of Yang et al. [29, 30], test structures based on scan chains can be easily used to mount attacks against cryptographic devices. In order to prevent such attacks, test structures of security-related devices are typically deactivated after successfully testing the chip (e.g. by blowing a fuse) or even totally removed by cutting them off [11].

An alternative to scan-chain approaches are built-in self tests (BISTs). The NIST suggests to use BISTs instead of scan-chains for cryptographic devices [14]. For a BIST, necessary test data and test cases are generated within the evaluated chip. The only information that is returned after conducting the tests is whether all tests have been passed or not. This is advantageous from a security point of view but comes at cost of a lower fault-detection rate, since comprehensive tests as with scan-chain approaches are not possible. Moreover, generating test data within the chip causes significant hardware overhead.

5.8 Hardware Trojans

Test structures implemented by a designer to be able to verify the correct functionality of an IC after production, may unintentionally weaken or bypass security measures implemented on that device. In contrast, hardware Trojans describe security-threatening hardware structures in ICs, intentionally implemented at some point in the fabrication chain. For cost reasons, more and more semiconductor companies outsource the chip fabrication process to cheaper facilities. Hence, hardware Trojans may be implemented without the designers' knowledge or notification (assuming that the designer is not an adversary). Moreover, they may remain undetected even if the designer receives the ICs from the wafer factory and performs conventional functionality tests, as hardware Trojans may or may not be activated as soon as the IC is powered up. Once activated, the malicious actions a hardware Trojan may perform are incredibly powerful:

- simply shut down the device the Trojan is running on or disable connected devices,
- disable security mechanisms in a system,
- transmit critical data via various interfaces (e.g. radio-frequency emission),
- open a backdoor for an adversary and provide access to a system,
- or bypass implemented hardware countermeasures against side-channel attacks.

Wang et al. [26] classified hardware Trojans into three classes according to their *physical*, *activation*, and *action* characteristics. The *physical characteristics* describe how the Trojan is introduced in a digital circuit (addition/deletion of cells or modification of existing cells), the size and the location of the Trojan (how many cells are involved), as well as if the insertion of the Trojan entails significant changes of the physical layout of the digital circuit. The *activation characteristics* describe whether the Trojan is "always-on" or has to be activated externally, e.g. via antenna, or internally. Wolff et al. [28] further divided internally activated Trojans into three categories based on their trigger behavior: rare value triggered, time triggered, and both value and time triggered. Wang et al.introduced three *action characteristics* that describe whether an activated Trojan modifies a function (addition/deletion of logic cells) or specification (e.g. modification of wires changes the timing specifications), or directly transmits critical information over various channels.

Preventing hardware Trojans is a very complex issue, as Trojans may be implanted in different phases, e.g. during the high level or hardware-level design phase of a system, during synthesis/place/route of hardware modules, or even during the fabrication process of an IC. One possible protection against hardware Trojans is a chip developer would have to somehow establish a chain of trust, starting at the designer, continuing with the hardware experts, up to the IC production facility and the package house.

Another possibility to detect hardware Trojans is based on SCA [2]. In this approach a few ICs from one IC family (produced with the same mask) are first subjected to sufficient I/O tests to verify all parts of the circuitry. During these tests, some side-channel signals are also collected to build a side-channel fingerprint. In a next step, the ICs are destructively reverse engineered in order to thoroughly ensure that these few samples are free of Trojans. All other ICs from the same family can then be checked by comparing the fingerprints. This example shows that security-threatening side-channel attacks can also be used to some degree to detect hardware Trojans in ICs.

5.9 Conclusion and Summary

Implementing efficient, secure, and reliable ICs is a highly sophisticated task that provides manifold possibilities, but requires to be performed with great care. In this chapter we have shown that a hardware designer has almost indefinite possibilities to basically add specific functionality to an IC as well as to optimize the implementation to reach various design goals. By means of several examples of cryptographic hardware implementations we have illustrated the degrees of freedom that are at a designer's disposal. We also pointed out the importance of performing tests and simulations throughout the whole design cycle in order to minimize the possibility of errors as well as to decrease the effort if an error occurs during the design phase.

With regard to security-threatening side-channel attacks we described the fundamental characteristics of modern CMOS circuits and pointed out the reason for the vulnerability of ICs against such attacks by means of a conventional CMOS inverter. We discussed basic approaches for implementing countermeasures against side-channel attacks and introduced some particular countermeasures in more detail. We also pointed out the possibility to verify the effectiveness of various types of countermeasures to some degree in early design phases. This minimizes the costs as well as the effort to perform changes in the implementation.

We combined the topics of efficient hardware implementations, cryptographic ISEs, and hardware countermeasures against side-channel attacks and presented a sophisticated concept with custom instructions and countermeasures on a modern SPARC V8-compliant 32-bit processor platform. It turned out that such a concept benefits from both the efficiency and compactness of ISEs as well as the security gain achieved by a secure logic style.

In a last part of this chapter, we contrasted the testability with the security of ICs. It has been shown that test structures implemented by designers to be able to comprehensively verify the correct functionality of a design after production may lead to significant security leaks. We also discussed the insertion and some possible effects of hardware Trojans, which represent a worrying yet interesting and currently evolving research topic.

References

1. Aeroflex Gaisler. The Aeroflex Gaisler Website. http://www.gaisler.com/.
2. D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan Detection using IC Fingerprinting. In *IEEE Symposium on Security and Privacy (SP '07), Berkeley, Californie, USA, May 20–23 2007*, pages 296–310, 2007.
3. D. Canright and L. Batina. A Very Compact "Perfectly Masked" S-Box for AES. In *Applied Cryptography and Network Security - ACNS 2008, New York, USA, June 3–6, 2008, Proceedings*, volume 5037 of *Lecture Notes in Computer Science*, pages 446–459. Springer, 2008.
4. M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES Implementation on a Grain of Sand. *IEE Proceedings on Information Security*, 152(1):13–20, October 2005.
5. Gaisler Research. LEON2 Processor Users Manual. XST Edition. [Online] http://www.gaisler.com/doc/leon2-1.0.30-xst.pdf, July 2005. Version 1.0.30.
6. D. Gajski and R. H. Kuhn. New VLSI Tools - Guest Eidtors' Introduction. *IEEE Computer*, 16(12):11–14, 1983.
7. A. Hodjat and I. Verbauwhede. Interfacing a High Speed Crypto Accelerator to an Embedded CPU. In *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems, and Computers, 2004,* volume 1, pages 488–492. IEEE, November 2004.
8. H. Kaeslin. *Digital Integrated Circuit Design - From VLSI Architectures to CMOS Fabrication.* Cambridge University Press, 2008. ISBN 978-0-521-88267-5.
9. P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18–22, 1996, Proceedings,* number 1109 in Lecture Notes in Computer Science, pages 104–113. Springer, 1996.
10. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15–19, 1999, Proceedings,* volume 1666 of *Lecture Notes in Computer Science,* pages 388–397. Springer, 1999.
11. O. Kömmerling and M. G. Kuhn. Design Principles for Tamper-Resistant Smartcard Processors. In *Proceedings of the 1st USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, Illinois, USA, May 10–11, 1999,* pages 9–20, McCormick Place South, May 1999. USENIX Association. ISBN 1-880446-34-0.
12. S. Mangard, M. Aigner, and S. Dominikus. A Highly Regular and Scalable AES Hardware Architecture. *IEEE Transactions on Computers,* 52(4):483–491, April 2003.
13. S. Mangard, T. Popp, and B. M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In A. Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14–18, 2005, Proceedings,* volume 3376 of *Lecture Notes in Computer Science,* pages 351–365. Springer, February 2005.
14. National Institute of Standards and Technology (NIST). FIPS PUB 140–1: Security Requirements for Cryptographic Modules, 1994. [Online] http://www.itl.nist.gov/fipspubs/.
15. National Institute of Standards and Technology (NIST). FIPS-197: Advanced Encryption Standard, November 2001. [Online] http://www.itl.nist.gov/fipspubs/.
16. T. Popp, M. Kirschbaum, T. Zefferer, and S. Mangard. Evaluation of the Masked Logic Style MDPL on a Prototype Chip. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10–13, 2007, Proceedings,* volume 4727 of *Lecture Notes in Computer Science,* pages 81–94. Springer, September 2007. ISBN 978-3-540-74734-5.
17. T. Popp and S. Mangard. Masked Dual-Rail Pre-Charge Logic: DPA-Resistance without Routing Constraints. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29–September 1, 2005, Proceedings,* volume 3659 of *Lecture Notes in Computer Science,* pages 172–186. Springer, 2005.

18. J. M. Rabaey. The SPICE Home Page. http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/.
19. J. M. Rabaey. *Digital Integrated Circuits - A Design Perspective.* Electronics and VLSI Series. Prentice Hall, 1st edition, 1996. ISBN 0-13-178609-1.
20. P. Schaumont and K. Tiri. Masking and Dual-Rail Logic Dont Add Up. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10–13, 2007, Proceedings,* volume 4727 of *Lecture Notes in Computer Science,* pages 95–106. Springer, September 2007.
21. D. Suzuki, M. Saeki, and T. Ichikawa. Random Switching Logic: A New Countermeasure against DPA and Second-Order DPA at the Logic Level. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences,* E90-A(1):160–168, 2007. ISSN 0916–8508.
22. S. Tillich and J. Großschädl. Instruction Set Extensions for Efficient AES Implementation on 32-bit Processors. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10–13, 2006, Proceedings, volume 4249 of Lecture Notes in Computer Science,* pages 270–284. Springer, 2006.
23. S. Tillich and J. Großschädl. Power-Analysis Resistant AES Implementation with Instruction Set Extensions. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10–13, 2007, Proceedings,* volume 4727 of *Lecture Notes in Computer Science,* pages 303–319. Springer, September 2007.
24. S. Tillich, M. Kirschbaum, and A. Szekely. SCA-Resistant Embedded Processors - The Next Generation. In C. Gates, M. Franz, and J. P. McDermott, editors, *26th Annual Computer Security Applications Conference (ACSAC 2010), 6–10 December 2010, Austin, Texas, USA,* pages 211–220. ACM Press, 2010.
25. K. Tiri and P. Schaumont. Changing the Odds against Masked Logic. In E. Biham and A. M.Youssef, editors, *Selected Areas in Cryptography, 13th International Workshop, SAC 2006, Montreal, Quebec, Canada, August 17–18, 2006, Revised Selected Papers,* volume 4356 of *Lecture Notes in Computer Science,* pp. 134–146. Springer, 2007. [Online] http://rijndael.ece.vt.edu/schaum/papers/2006sac.pdf.
26. X. Wang, M. Tehranipoor, and J. Plusquellic. Detecting Malicious Inclusions in Secure Hardware: Challenges and Solutions. In M. Tehranipoor and J. Plusquellic, editors, *Hardware-Oriented Security and Trust (HOST 2008), Anaheim, CA, June 9 2008, Proceedings,* pages 15–19, 2008.
27. N. H. E. Weste and D. Harris. *CMOS VLSI Design—A Circuits and Systems Perspective.* Addison-Wesley, 3rd edition, May 2004. ISBN 0-321-14901-7.
28. F. G. Wolff, C. A. Papachristou, S. Bhunia, and R. S. Chakraborty. Towards Trojan-Free Trusted ICs: Problem Analysis and Detection Scheme. In *Design, Automation and Test in Europe (DATE), 10–14 March, 2008,* 2008.
29. B. Yang, K. Wu, and R. Karri. Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard. In *Proceedings of the International Test Conference on International Test Conference,* CCS '05, pages 139–146, New York, NY, USA, 2005. ACM.
30. B. Yang, K. Wu, and R. Karri. Secure Scan: A Design-for-Test Architecture for Crypto Chips. *IEEE Trans. on CAD of Integrated Circuits and Systems,* 25(10):2287–2293, 2006.
31. P. Yu and P. Schaumont. Secure FPGA circuits using controlled placement and routing. In *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis, Salzburg, Austria, September 30 - October 5, 2007,* pages 45–50. ACM Press, September 2007. ISBN 978-1-59593-824-4.