Early Feedback on Side-Channel Risks with Accelerated Toggle-Counting

Zhimin Chen
Electrical and Computer Engineering Dept.
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
chenzm@vt.edu

Patrick Schaumont
Electrical and Computer Engineering Dept.
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
schaum@vt.edu

***Abstract*—Early detection of side-channel leakage in the design of a digital crypto-circuit is as important as getting the design functionally correct. Currently, side-channel leakage is confirmed by measuring actual prototypes, or by detailed SPICE-level simulations of the hardware model. However, this feedback does not help the designer: it comes either too late (after the implementation), or else it has unreasonable simulation-time requirements. In this paper, we present a design method to provide early feedback on side-channel leakage based on toggle-counting. We show how to abstract the low-level circuit effects, such as glitches and routing imbalances, into high-level simulation models that can be toggle-counted. Furthermore, we propose an acceleration method for fast toggle-counting using reconfigurable hardware. The proposed methods accelerate the evaluation of side-channel leakage by a factor of 10^5 over comparable SPICE-level simulations.**

Keywords- Side-Channel Attack; High-level Simulation; High-level Glitch Model; Fast Toggle Count

I. INTRODUCTION

The design of digital crypto-circuits is a complex design activity, especially when side-channel leakage needs to be minimized. Common problems in digital design can be solved with tools and methods. For example, simulators aid the functional verification of a design, and timing analyzers check the projected performance of the implementation. In contrast, there are no tools available to help a designer evaluate security risks in the *early* stages of a design flow. In this paper, we propose a design method to qualitatively evaluate the risk for side-channel leakage. The method optimizes simulation speed without abstracting away the critical details from the design under consideration.

Indeed, side-channel leakage is commonly associated with the low-level implementation effects in a crypto-circuit, such as timing effects, glitches, routing imbalance, and so forth. It is therefore assumed that accurate observation is only possible at low abstraction levels. When a simulation tool abstracts these implementation effects away, it can hide the presence of a side-channel leak [1].

The evaluation of side-channel leakage requires an accurate evaluation of a circuits' power consumption. There are two common ways to do this. The first is to measure power on the actual implementation, and the second is to perform SPICE-level simulation of a circuit model. Both approaches are accurate and useful for side-channel analysis, but each of them suffers from a specific weakness. Using power measurements on the implementation suffers from *observability*; often a side-channel leak is found which cannot be explained until detailed study of a simulation model of the chip is done [2]. SPICE-level simulation models on the other hand provide high observability, but they are very slow because their *accuracy* is much higher than what is needed for common types of side-channel analysis. For example, it is known that glitches can cause side-channel leakage. However, one does not need to know the exact shape of a glitch waveform to flag them as a side-channel risk.

A design method that provides early feedback on side-channel risks therefore involves a trade-off among three factors: (simulation) speed, observability, and accuracy. Compared to power measurements on the actual implementation, and SPICE-level simulation, we seek a method that gives high speed as well as high observability. The speed in our method comes from modeling the entire circuit as a cycle-based simulation. We perform per-clock-cycle toggle counting to estimate the power dissipation of the resulting model. Of course, cycle-based simulation removes low-level timing effects (e.g. glitches, late-arrival of signals) as well as analog loading effects (e.g. routing capacitance). Therefore, we capture those low-level effects into higher-level toggle-counting models. In this paper, we will demonstrate that the proposed abstractions are still able to analyze side-channel leakage. Furthermore, we demonstrate a hardware-accelerated version of our simulation method on FPGA, which offers a simulation speedup of 10^5 over the original HSPICE-based simulation.

The remainder of this paper is organized as follows. In Section II, we review related work on estimation of side-channel leakage in digital design. In Section III we review the proposed design methods, emphasizing the abstraction of glitches into cycle-based simulation models. We also describe a hardware acceleration technique for the resulting simulation models. Section IV describes the design flow, and Section V

978-1-4244-4804-3/09/$25.00 © 2009 IEEE

demonstrates it using a design example. We conclude in Section VI.

II. PREVIOUS WORK

A very common verification method to evaluate side-channel resistance is based on simulated DPA using SPICE models. The objective of such simulations is to investigate the dependency of a circuit's power consumption on a specific input bit of a gate, or on the specific arrival time of signals at the input of a gate [3][4]. Because SPICE models are very compute-intensive, these simulations are restricted to systems of only a few logic gates at a time. In contrast, we seek an approach where a designer can analyze an entire module directly, without analyzing and decomposing that module first into lower-level, more detailed simulation models.

Suzuki and Saeki presented leakage models for CMOS Logic Circuits based on the output signal transition probability for each individual gate [5][6]. Their simulation models are constructed at the logic-level and execute as event-driven simulations or as FPGA prototypes. In our work, we try to abstract the simulation further as cycle-level simulations that can be emulated on FPGA. Our key observation is that even event-driven simulation may not be fast enough to extract the transition probability of each gate. For example, Mangard presents the side-channel analysis of a masked SBOX [7]. Because of glitches inside of this SBOX, he can construct a successful DPA after 100,000 input vectors, which, from our experiment, takes almost 1 hour. Further, this is only a tiny subset of the possible transitions for this SBOX. Since it has 12 mask bits and 8 input bits, the exhaustive simulation of all possible transitions would take $(256x4096)^2 = 1.09 \times 10^{12}$ input vectors! Therefore, event-driven logic simulation is still inconvenient. We also notice that some other simulation methods or tools can be used for DPA analysis, such as SWAN [8], PrimePower, Nanosim and so on. These are faster than SPICE. However, similar to event-based logic simulation, their performance is still not satisfying in front of a huge amount of simulations.

Of course, accuracy is clearly a concern when using simulation models in side-channel analysis, and Tiri has pointed out that the proper simulation model for power-based side-channel analysis is an open question [8]. In this context, it is useful to define the nature of the *qualitative* answer on side-channel leakage that we seek. Since we use simulation models at high abstraction level, we will not be able to guarantee that the circuit implementation of those models will be absolutely free of side-channel leakage. However, we target to identify circuits that are very likely to cause side-channel leakage in their implementation. Thus, we seek a yes-or-no answer to the question: will this design exhibit exploitable leakage when implemented? We do *not* attempt to quantify the exact amount of side-channel leakage, nor do we want to prove that a circuit will be free of side-channel leakage. The yes-or-no answer for side-channel leakage is similar to the timing-constraint checking during digital synchronous design. In that case a designer is primarily interested in knowing if a designs' critical path will meet the design constraint, while an exact quantification is not yet needed.

III. DESIGN METHODS

In this section, we describe how to abstract low-level circuit-effects into cycle-based simulations. We also provide an emulation method of the resulting simulations in FPGA.

Cycle-based logic simulation partitions the behavior of a logic design in simulation steps of one clock period. Each clock period has two phases: (1) evaluation of combinational logic, and (2) update of registers. Because all registers are updated together in a single atomic simulation step, cycle-based simulation cannot capture circuit effects shorter than a single clock period. Furthermore, a cycle-based simulator does not handle the load differences of different wires in a circuit. A wire carries either a logic-1 or else a logic-0, and each logic transition requires a unit amount of energy. On the plus side, a cycle-based simulation is very simple, and therefore it is very fast.

A. Abstraction of low-level circuit features

To enable the observation of the glitches based on cycle-based simulation, we abstract the continuous timing delay of the circuits to a unified delay for each logic gate (or Look-Up Table). Based on this abstraction, we then modify the original simulation model into a glitch-aware cycle-based simulation model using three steps:

1) Suppose the longest logic depth of the circuit is n gates, then we modify the original flip-flops such that they are enabled one out of n simulation clock cycles;

2) We insert a new flip-flop (always enabled) after every combinational gate to add unit-delay to the output;

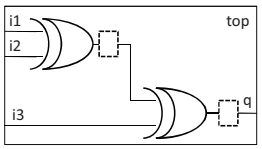
3) We prolong the switching rate of the input stimulus from 1 to n simulation clock cycles.

We define the original clock cycle as the '*logic cycle*'. According to the above modifications, the *logic cycle* is divided into *n* simulation clock cycles. To differentiate this new clock cycle from the previous one, we define it as '*simulation step*'. Obviously, because one *simulation step* is *1/n* of one *logic cycle*, this gives us a chance to observe the glitches with cycle-based simulation. An example will be given in the next section.

Although the unified timing delay model is different from the delay found in an actual circuit, it is much better than the ideal gate model without timing delay. Moreover, since we only look for a qualitative feedback, our delay model does not need to be exact. Furthermore, an improvement can be done by adding different numbers of flip-flops after the combinational gates according to their delay difference. This brings higher accuracy but increases the simulation time. In this paper, only the unified timing delay model is considered.

To enable the load capacitance observation, we can substitute the logic-1-logic-0 model to a weighted-load model. The weight values can be obtained either based on the fan-out of the gates, or else from a random distribution. In this paper, we only consider the glitch models. The design method is however not restricted to glitches, and can be extended to the weighted load model.

```
1.  $option "profile_toggle_upedge"
2.  $option "toggle_include top"
3.
4.  dp xor(in i1,i2:ns(1); out q:ns(1)) {
5.    q = i1 ^ i2;
6.  }
7.
8.  dp xor_g(in i1,i2:ns(1); out q:ns(1)) {
9.    reg c : ns(1); // unit delay
10.   c = i1 ^ i2;
11.   q = c;
12. }
13.
14. dp dut_xor1:xor_g
15. dp dut_xor2:xor_g
16.
17. dp top(in i1,i2,i3:ns(1);out q:ns(1)) {
18.   sig n : ns(1);
19.   use dut_xor1(i1,i2,n);
20.   use dut_xor2(n,i3,q);
21.   always { }
22. }
23.
24. dp test {
25.   sig i1, i2, i3, q : ns(1);
26.   use top(i1, i2, i3, q);
27.   always {
28.    // apply stimuli
29.    ...
30.    $display("Cycle= ",       $cycle,
31.             "H-distance= ", $toggle,
32.             "H-weight= ",   $ones);
33. }
```



Listing 1: Toggle Counting in GEZEL

B. *Toggle counting based on software logic simulation*

Our software-based simulations of toggle-counting models are based on GEZEL models [10]. Listing 1 illustrates the main features of the toggle counting support using a small network with two `xor` gates (shown in the inset of Listing 1). GEZEL supports structural hierarchy. `xor` gates can be captured as hardware modules (Lines 4-6), which are instantiated later inside of another module (Lines 19-20). For glitch simulation, we need gates with unit-delays. This is modeled using a register at the output of the gate, as shown in the `xor_g` module (Line 8-12). For easy reconfiguration between the original cycle-based simulation and the glitch-enabled simulation, we express the module-under-test in terms of generic gates, such as `dut_xor1`, `dut_xor2` in Listing 1. The simulation is then configured by choosing a leaf-node implementation for each generic gate (Lines 14-15).

Toggle-counting is enabled through simulation directives (Lines 1-2). The net-transition types to be monitored can be selected: `upedge` (0→1), or `bothedge` (0→1 and 1→0). The simulation directives can also restrict toggle counting to a specific module or a group of modules in the design. During simulation, the testbench (Lines 24-33) can query the Hamming distance (Line 31) or the Hamming weight (Line 32)

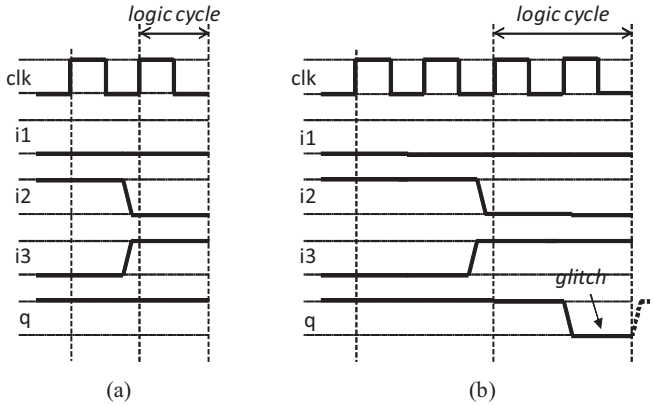


Figure 1: Simulation of Listing 1 (a) without glitch modeling and (b) with glitch modeling

of all nets within a specified module using the `$toggle` and `$ones` directive respectively.

An example of the simulation for the two `xor` gates from Listing 1 is illustrated in Figure 1. When the `xor` cells are used, each simulation step in GEZEL will simulate a single logic clock cycle of the circuit. When the `xor_g` cells are used, each gate has a unit delay of one simulation step. As shown in Figure 1b, a logic cycle can extend over multiple simulation steps. Glitches appear as transitions of at least one simulation step, but shorter than a logic cycle. Since the simulation toggle-counting will include all transitions, glitches will be included in the overall toggle count. To find the number of toggles per logic cycle, the toggle-counts of all simulation steps within the same logic cycle are accumulated.

The number of simulation steps per logic cycle must be larger than the logic depth of the circuit to ensure that the outputs have reached a stable value. The logic depth of the `top` cell in Listing 1 is two gates, so that we need two simulation steps per logic cycle. In practice, we select a safe upper-bound that ensures all glitch effects have propagated to the output of the circuit. For example, for an AES SBOX synthesized to multilevel logic gates, we found 32 simulation steps to be a safe upper-bound.

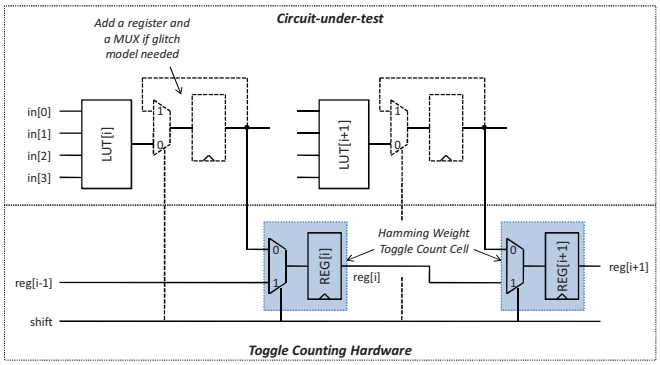


Figure 2: Hardware Simulation with Hamming Weight Toggle Counting Cells.

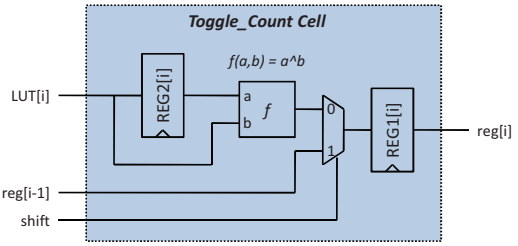


Figure 3: Structure of a Hamming Distance Toggle Count Cell.

C. *Toggle counting based on hardware logic simulation*

Because side-channel analysis involves a large number of experiments, even simulation at high level may not be fast enough. Therefore, we investigated a method to use configurable hardware (FPGA) to further accelerate the simulation. In this section, we show how to implement the above software-based simulation in an FPGA.

The basic idea is that we collect the outputs of all logic cells of interest using a scan chain, as shown in Figure 2. This scan chain is created by attaching 'toggle-counting' cells to the logic cells of the circuit-under-test. When glitch information is needed, we also include a unit-delay flip-flop at the output of every logic gate in the original circuit. Both of these transformations are easy to formulate starting from the original netlist.

In every software simulation step, the circuit in Figure 2 will alternate between two phases, one for calculation and sampling (sampling phase, shift=0) and the other for shifting (shifting phase, shift=1). Every sampling phase lasts for 1 FPGA clock cycle and is followed by a shifting phase.

If there are *s* added counting cells and only one scan chain, then the shifting phase lasts at least for *s* FPGA clock cycles. At the end of the scan chain, we add a ones-counter to count the number of logic-1s obtained in the sampling phase. After *p* simulation steps in the shifting phase, the counter value corresponds to the Hamming Weight result. Therefore, one software simulation step consists of at least *p+1* FPGA clock cycles.

With the same method, we can also obtain the toggle counts by replacing the counting cells with the Hamming Distance toggle counting cells as shown in Figure 3. The toggle counting cell employs a second flip-flop REG2 to store the output value in the previous simulation step. In the sampling phase, with the logic function *f*, we compare the previous value with the current value to generate the Hamming distance value and store it in REG1. After that, the circuit switches to shifting phase for toggle counting.

Although one software simulation step is expanded to *p+1* FPGA clock cycles, the hardware simulation is still much faster than the software simulation. This speed up comes from the parallel logic computation and sampling. When the circuit under analysis is large, *p* may become very big. However, the scan-chain method is very easy to parallelize: simply cut the scan chain into several smaller ones, and perform ones-counting in each of them. Finally, we add up the values obtained by each counter and form the final result.

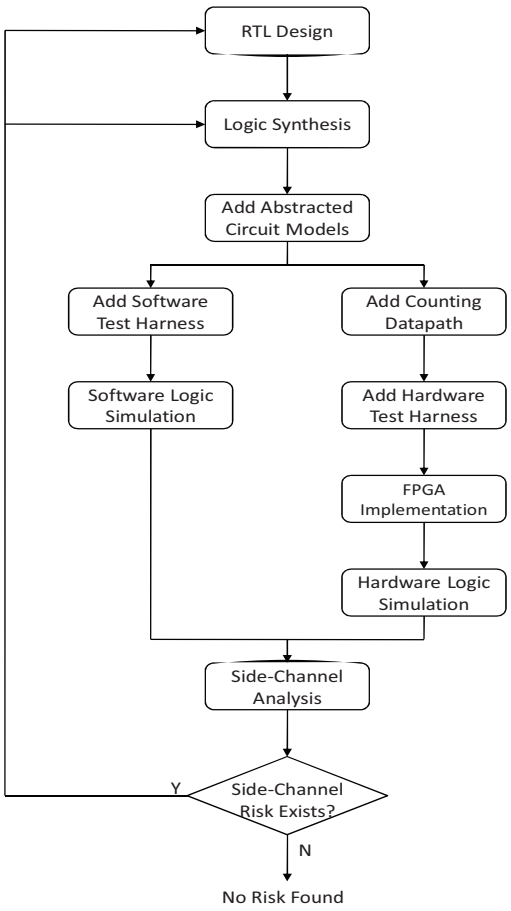


Figure 4: Design Flow with Early Feedback of Side-channel Risks

Another concern may be that this method is too resource intensive. In the results section, we will show that this remains reasonable for the case of FPGA, and that the resulting speedup more than justifies the use of FPGA emulation.

IV. Design Flow

Figure 4 shows a design flow for early feedback of the Side-Channel risks. We assume that the logic function of both the RTL design and the synthesized netlist is correct and only focus on the security features of the circuit design. Based on the synthesized netlist, we first add the abstracted circuit models, such as unit-delay model, weighted load model and so on. The resulting netlist can be tested with software logic simulation after the integration of a suitable test harness. On the other hand, we can also insert the netlist with the Hamming weight or toggle counting hardware and a test harness for hardware logic simulation. After FPGA implementation, hardware logic simulation can be performed. Both the software and hardware logic simulations generate the same Hamming weight or the toggle count results, which will go through standard Side-Channel Analysis techniques, such as Differential Power Analysis (DPA) or Correlation-based Power

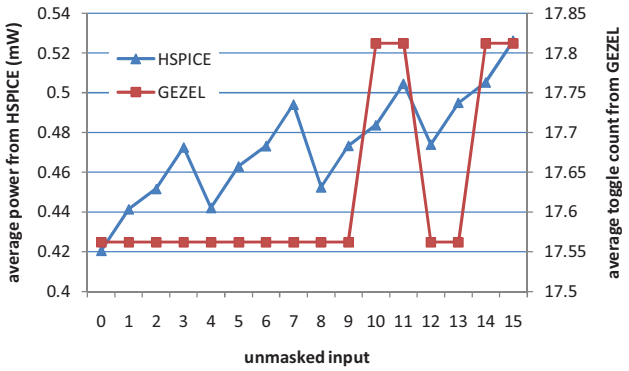


Figure 5: Both GEZEL and HSPICE show that the average power is dependent on the unmasked input: there is risk of Side-channel leakage.

Analysis (CPA). Finally, if the analysis reports a risk, further refinement should be done either with the RTL design (e.g. to improve the algorithm) or the logic synthesis (e.g. to protect the masking elements). Otherwise, the synthesized netlist passes the test. Although passing the test does not imply that the implementation will be totally free from side-channel leakage, this early feedback is still valuable to designers.

V. Experimental Results

In this section, we will present and compare the results obtained from SPICE-level simulations, software logic simulation, and hardware logic simulation. We will discuss two different circuits: a perfectly masked multiplier and a perfectly-masked S-BOX [11]. For each of these circuits, it is known that they leak side-channel information through glitch effects. Hence, they are an adequate target to validate our approach.

A. Comparison between software logic simulator and HSPICE

The circuit under test in the first step is a masked GF(2^2) multiplier [11] from a masked AES SBox. The study in [2] has shown that a masked GF(2^2) multiplier is a possible leakage source because of glitches. We perform both SPICE-level simulation and software logic simulation to get the power estimations.

We used HSPICE for SPICE-level simulations on the netlist with the TSMC 0.18µm technology (transistor level BSIM3). We switch the inputs from all 0 to every possible input value. Each circuit has two 2-bit data inputs, and 3 two-bit masking inputs. Accordingly, 1024 experiments are needed to cover the entire input space. Finally, the average power in each experiment is obtained and indexed according to the hamming weight of the unmasked data input. If the resulting circuit is perfectly masked, then the power consumption should not reveal any information on the unmasked input [12]. Hence, all of the average power consumption values should be equal.

The software logic simulation (GEZEL) integrates the abstracted glitch model and takes 16 as the logic depth and does the same task as above. Thus, 16384 simulation steps are performed. Finally, the average toggle count in each logic cycle is obtained.

Table 1: Hardware resource results

	LUT	***Flip-Flop***
original netlist	246	0
netlist with HW counting cell	497	247
netlist with toggle counting cell	502	494
netlist with HW counting cell and glitch model	494	496
netlist with toggle counting cell and glitch model	494	744

Both the above two simulations run on an Intel Xeon 3GHz workstation with a 2GB memory. The HSPICE takes 522.46 seconds while GEZEL only needs 1.16 seconds. We group the obtained power according to the unmasked inputs to see the dependency between the power and the unmasked inputs, shown in Figure 5.

From Figure 5, we can see both simulations show a dependency between the unmasked input and the power value, which tells us that the Side-Channel leakage exists. Because of the abstracted glitch model and simplified dynamic power model, the result from GEZEL simulation is different from the one from HSPICE. However, this difference does not prevent us obtaining the qualitative feedback.

To summarize, the first step demonstrates that the logic simulation is able to give a qualitative feedback on the Side-Channel risks. Moreover, the software logic simulation is 450 times faster than HSPICE simulation.

B. Comparison between software and hardware logic simulations

Since both the software and hardware logic simulations are functionally identical, their results are the same. So we only compare their difference in terms of execution speed.

The circuit under test in this case is a masked AES SBox [11]. After synthesis with Xilinx ISE, 246 LUTs are used. The upper bound of the logic depth is 32. Going through the design flow, we obtain the simulation setups for both software and hardware simulations. The test harness in the FPGA (Virtex II pro, xc2vp30) system is a control shell attached to a MicroBlaze processor with a FSL connection. The whole FPGA system runs at 100MHz.

In both of these two simulations, 131,072 experiments were performed with the glitch model. The software logic simulation takes 59 minutes and 51 seconds while hardware simulation takes $1.329*10^9$ cycles (13.29 seconds). The hardware acceleration speed up is 277. Considering the speed up of software logic simulation over HSPICE is 450, the hardware logic simulation runs approximately *124,650* times faster than the HSPICE simulation.

From the resource usage in Table 1, we see that the largest area cost comes from the netlist with toggle counting cell and glitch model. The numbers of LUTs and Flip-flips are 2 times and 3 times of the LUT number in the original netlist. Note however that the resource cost listed here is not part of the actual design, and is only needed for simulation acceleration.

VI. CONCLUSION

In this paper we demonstrated how toggle-counting can be used as a side-channel evaluation mechanism, and we demonstrated a mechanism to improve its performance. This work is motivated by the need to assist the crypto-engineer with useful tools that can be applied during the early stages of the design flow. We are currently applying the technique to large cryptographic modules to demonstrate that it is scalable. We are also evaluating how to address other low-level circuit features, including load-imbalance in dual-rail design.

REFERENCES

[1] K. Tiri, I. Verbauwhede, "Simulation Models for Side-Channel Information Leaks", Design Automation Conference (DAC 2005), pp. 228-233, June 2005.

[2] S. Mangard, L. Schramm, “Pinpointing the side-channel leakage of masked AES hardware implementations,” International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2006), LNCS vol 4249, 76-90, Springer, 2006.

[3] E. Menendez, K. Mai, “A High Performance, Low Overhead, Power Analysis Resistant, Single-Rail Logic Style, ” Proc. of the IEEE International Workshop on Hardware-Oriented Security and Trust, 33-36, 2008.

[4] S. Guilley, P. Hoogvorst, F. Flament, R. Pacalet, Y. Mathieu, “Secured CAD Back-End Flow for Power-Analysis Resistant Cryptoprocessors,” IEEE Design and Test of Computers, 24(6):546-555, IEEE, 2007.

[5] D. Suzuki, M. Saeiki, T. Ichikawa, “DPA Leakage Models for CMOS Logic Circuits,” International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005), LNCS vol 3659, 366-382, Springer, 2005

[6] M. Saeki, D. Suzuki, T. Ichikawa, “Leakage Analysis of DPA Countermeasures at the Logic Level,” IEICE Trans. Fundamentals, E90-A, 169-178, January 2007.

[7] S. Mangard, N. Pramstaller, E. Oswald, “Successfully Attacking Masked AES Hardware Implementations,” International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005), LNCS vol 3659, 157-171, Springer, 2005.

[8] K. Tiri, “Side-channel Attack Pitfalls,” Proc. of the ACM-IEEE Design Automation Conference (DAC 2007), 15-20, ACM-IEEE, 2007.

[9] M. Badaroglu, G. Van der Plas, P. Wambacq, S. Donnay, G. G. E. Gielen, and H. J. De Man, "SWAN: High-level Simulation Methodology for Digital Substrate Noise Generation", IEEE Transactions on VLSI Systems, Vol. 14, no. 1, 2006.

[10] P. Schaumont, I. Verbauwhede, "A Component-based Design Environment for Electronic System-level Design," IEEE Design and Test of Computers, special issue on Electronic System-Level Design, 23(5), pp. 338-347, September-October 2006.

[11] E. Oswald, S. Mangard, N. Pramstaller, and V. Rjmen, “A Side-Channel Analysis Resistant Description of the AES S-box,” Fast Software Encryption (FSE 2005), LNCS vol 3557, 413-423, 2005.

[12] J. Blőmer, J, Guajardo, and V. Krummel, “Provably Secure Masking of AES,” Selected Areas in Cryptography (SAC 2005), LNCS vol 3357, 69-83, 2005.