Automated Design of Cryptographic Devices Resistant to Multiple Side-Channel Attacks

Konrad Kulikowski, Alexander Smirnov, and Alexander Taubin

Department of Electrical and Computer Engineering, Boston University,
8 Saint Mary's Street, Boston, MA 02215, USA,
{konkul, alexbs, taubin}@bu.edu

Abstract. Balanced dynamic dual-rail gates and asynchronous circuits have been shown, if implemented correctly, to have natural and efficient resistance to side-channel attacks. Despite their benefits for security applications they have not been adapted to current mainstream designs due to the lack of electronic design automation support and their non-standard or proprietary design methodologies. We present a novel asynchronous fine-grain pipeline synthesis methodology that addresses these limitations. It allows synthesis of asynchronous quasi delay insensitive circuits from standard high-level hardware description language (HDL) specifications. We briefly present a proof of concept differential dynamic power balanced micropipeline library cells that are approximately 6 times more balanced than the best (differential dynamic) cells designed using previous balancing methods. An implementation of the Advanced Encryption Standard based on these balanced cells and synthesized using our tool flow shows a 6.6 times throughput improvement over the synchronous automatically pipelined implementation using the same TSMC 0.18μm technology synthesized from the same HDL specification.

1 Introduction

Strong cryptographic algorithms have been designed to withstand rigorous cryptanalysis. However, if the overall cryptographic system is considered, including the physical implementation, the strong notions of security are far from guaranteed. Numerous attacks have been developed that exploit physical properties of implementations and information leaked through *side channels*, i.e. channels other than the data channel. By exploiting the information leaked through side-channels an attacker, with the help of statistical methods can quickly compromise the system. Side-channel attacks (and especially a combination of several such attacks) are often much more powerful than classical cryptanalysis.

In this paper we present a design methodology and a practical commercial quality Electronic Design Automation (EDA) flow which addresses the current practical and physical limitations. Our methodology provides tool support for the complete design cycle of secure cryptographic hardware which is capable of eliminating practically all sources of non-invasive side-channel information while allowing for very high performance of the implementation as well as low design time. It is based on asynchronous fine-grain pipelining with power-balanced cells to combine high performance with the best available power analysis resistance and excellent fault attack countermeasures. Our tool flow is based on off-the-shelf commercial EDA tools and does not require any specialized asynchronous design training or modifications of the original specification. Our flow is customizable through library approach to use various micropipeline implementations. Area and performance can be tuned as in other commercial quality synchronous synthesis flows.

The benefits of our methodology stem from two critical differences from previous implementations and are a direct consequence of the fine-grain structure of the final implementations. Fine-grain micropipeline cells are suitable for automatic ASIC synthesis from HDL specifications. Efficiently implemented using dynamic differential dual-rail circuitry micropipelines make it possible to automate normally custom power ballanced circuits design.

Using our methodology we have designed a complete hardware implementation of Advanced Encryption Standard (AES) [1]. Our AES implementation was synthesized for balanced library using our EDA tools. It combines differential power analysis (DPA) attack resistance with high performance. To satisfy the tight time-to-market and ease of design constraints our EDA flow accepts standard HDL input behavior specifications. Our design flow is complete. It incorporates off-the-shelf industrial tools with our scripts and reimplementation engine.

In this paper we first (Section 2) review previous work and some weaknesses and limitations of existing approaches. In Section 2 we also shortly introduce the key concepts of asynchronous design. The next two sections describe the keys for the success or our approach. Section 3 describes the main idea behind asynchronous fine-grain pipelining, implementation basis and automated design flow which allows asynchronous design using standard methodologies currently used in practically all automated synchronous design flows. Section 4 describes some details of the design and considerations of a dedicated balanced library cells for asynchronous fine-grain pipelining which are easily customizable and can be adapted to many existing balanced gate styles. Section 5 shortly describes performance characteristics of AES implementation and Section 6 presents short conclusions and future tasks.

2 Motivation

2.1 Dynamic logic and security

Some of the most promising methods for DPA resistance are based on specially designed balanced dynamic gates like those from [2]. Recent results on DPA resistance based on special power-balanced cells [2, 3] show a significant reduction in the power consumption fluctuations. Specially designed custom cells have great potential since instead of masking or hiding they remove power related sources of side-channel information that can be used for an attack.

However, most of gate-level approaches, such as those from [2, 3] have no countermeasures against glitch and fault-injection attacks and require additional protection. More importantly, since differential and dynamic (DD) approaches from [2, 3] require dynamic (domino) logic cell design. The usage of DD gates is limited to custom or semi-custom design that greatly limits the perceived universality of DD based circuitry. The following are two major reasons why EDA support of dynamic logic based design is very difficult for synchronous methodology [4, 5]. First, each synchronous dynamic gate requires a clock input and uses both levels of clock signal – it means that from the point of view of EDA tools each gate behaves like a flip-flop. Second, due to early/late arrival, charge sharing, clock distribution problems with small clocking granularity and uncertainty about worst case delay makes *static timing analysis* (STA) of dynamic circuits very problematic. STA is core part of any synchronous EDA approach. As a result no EDA tool support is available for synchronous design based of dynamic logic. As these problems make power balanced dynamic circuitry practically unavailable for rapid ASIC development the researchers resort to less secure (e.g. less balanced) but easier to implement solutions based on standard static non-balanced gate libraries (see e.g. motivation to use WDDL from [6]).

Our approach incorporates dynamic gate balancing techniques and methods with asynchronous design principles to address the timing and clock related problems associated with current and future balanced dynamic gate designs and to enable their use in automatic standard-cell based design flow.

2.2 Asynchronous Circuit Design

Many of the properties which many designs try to artificially add to synchronous designs are natural in some styles of asynchronous circuits. Some of the benefits previously noted include:

- Electromagnetic (or power) signature is strongly reduced by replacing a synchronous processor with an asynchronous one (no clock harmonics). Removing clock results in significantly flatter noise and electro magnetic interference (EMI) spectrum across the frequency domain (10dB drop according [7]).
- Absence of clock hardens triggering data detection at specific points of the data processing flow.
- With no clock glitch attacks are infeasible.
- In synchronous implementations, power supply fluctuations are used to force the circuit into an erroneous state allowing the use of differential fault analysis (DFA) attacks. Asynchronous circuits are much less sensitive to DFA attacks since the supply voltage drop gracefully slows down the circuit rather than leading to errors.
- Recent research suggests that asynchronous implementations have better resistance to power analysis and fault injection than synchronous counterparts. However, known implementations are still susceptible to information leakage both in power signature [8] and under fault injection. Contrariwise, balanced dual-rail domino with completion detection library - cell design that we chose

for implementation of asynchronous fine-grain pipelining eliminates a side-channel for DPA [8, 9].

– Comparison of electromagnetic analysis (EMA) results for synchronous and asynchronous implementations indicates that synchronous devices have data dependent EM emission, while non-pipelined asynchronous devices have data dependent timing visible with differential EMA (DEMA) [10].
– Asynchronous multi-dimensional (e.g. 3D) pipelined array architectures [11] can eliminate data dependent timing and thereby secure implementations against DEMA and differential timing analysis (DTA).

Various asynchronous design styles differ in the tradeoff between locality of timing assumptions and design cost (see e.g. [12]). *Quasi-delay-insensitive* (QDI) circuits [13] partition wires into critical and non-critical. Forks on critical wires are considered safe if they are isochronic – the skew is less than the minimum gate delay.

Universality and flexibility along with ease of design is a critical requirement necessary for the integration of any approach. QDI implementations appear to be the most appropriate – class of asynchronous circuits that can be synthesized automatically from large high-level behavior specifications. Return to zero handshaking protocol with dual-rail one-hot data encoding that switche the output from data to spacer and back regardless for every data portion is the most common QDI implementation. The most efficient QDI implementations are based on differential dynamic logic. That makes it easy to incorporate existing dynamic domino style power balanced structures in the QDI templates.

2.3 EDA support for Asynchronous Design

QDI based approach developed by TIMA group [14] is based on complex static library cells (built from basic gates like *C-elements* [15] etc.). These cells are not compatible with e.g. SABL [2]. In addition, TIMA tool flow [16] uses a non-standard language extension (channels) of HDL that require rewriting of design specifications.

Most importantly, none of known asynchronous EDA tools address fine-grain asynchronous dynamic logic pipelining which is of major importance for security and high performance. Fine-grain asynchronous pipelining seems to be the only way to move most promising DPA resistant (differential dynamic well balanced gates like SABL) into engineering practice since it seems to be the only way to provide EDA tool support for dynamic logic based styles.

In a summary, differential dynamic well balanced gates seem to be the best choice to design secure hardware resistant to side-channel attacks. Because of the time-to-market pressure without a solid EDA support any methodology for secure hardware design is likely to remain unused. QDI implementation methodology is able to play a key role by making dynamic cell libraries acceptable for EDA. Fine-grain asynchronous pipelining is a way to develop commercial quality tool support for QDI cell libraries. It becomes possible based on synchronous-to-asynchronous directed translation (SADT) approach. The main idea of SADT is to start from conventionally synthesized synchronous circuit, and directly replace the global clock network with a set of local handshake circuits. This way synthesis is performed by commercial synthesis tools originally developed for synchronous circuits. Since in dynamic logic each gate is a subject of clocking, fine-grain asynchronous pipelining by inserting local handshake control on the level of inter-gate communication (gate level pipelining) leads not only to convenient assimilation of differential dynamic balanced cell designs but also to high throughput solutions. In the next sections we explain how these necessary components lead to fine-grained structures and how they allow synthesis and other tool support.

3 Asynchronous micropipelines synthesis

Register Transfer Level (RTL) synthesis model simplified the clocked circuits' design and allowed design automation driving VLSI progress for more than a decade. Synchronous-to-asynchronous directed translation (SADT), we believe, is as important for asynchronous design automation as RTL for synchronous EDA. With RTL design dominating the industry SADT model is especially beneficial since (1) it offers support for existing specifications and (2) it is easily incorporated into contemporary design flow using the best available RTL synthesis engines. The handshake implementation and data channel organization is thereby hidden from the designer. Like in RTL it is customizable through a cell library approach.

Contrary to known approaches [16, 17] which use HDL for *micropipeline* [18] synthesis, our method is not an attempt to express asynchronous formal models in terms of HDL. Our synthesis flow uses an off-the-shelf RTL synthesis engine as a front-end to support regular HDL behavior specifications and the same engine as a back-end to provide support for the variety of netlist specification formats used by post-synthesis tools in ASIC design flow.

The main contribution of RTL model to EDA is based on a separation of optimization and timing (all sequential behavior is in an interaction between registers, all synthesis and optimization are only about combinational clouds). RTL model (Fig. 1a) is based on global synchronization and timing assumption (computations are complete in every stage before the next clock edge). During every clock cycle every latch undergoes two phases: pass and store. Master-slave flip-flop organization where master latch is clocked by one edge of clock signal and slave latch by the opposite edge prevents the register from being transparent at any given time. Similarly to pass and store of latches dynamic gates go through: *evaluate* and *precharge* (reset). These stages map to asynchronous four-phase handshake protocols [12] where the four phases are data request-acknowledge (evaluate) and request-acknowledge reset. (Fig. 1b).

In addition to separation of optimization and timing SADT model contributes separation of set and reset phases: for example each gate in Null Convention Logic (NCL) [19] is sequential but can be presented as combinational – separately in set and reset phases. As a result in SADT flow logic optimization

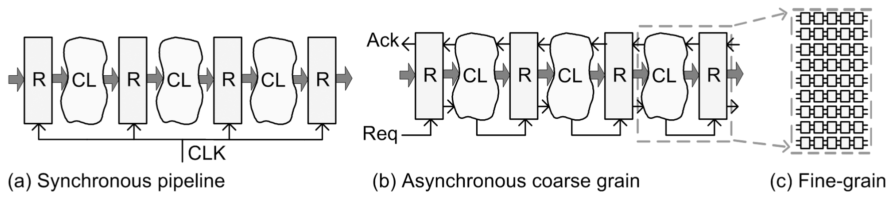


Fig. 1. Synchronous-Asynchronous Direct Translation: from synchronous (a) to de-synchronized (b) and fine-grain pipelined (c) circuits

remains separate from sequential behavior – the reason why SADT flows can be based on standard synchronous RTL compilers. Likewise sequential behavior synthesized in RTL remains the same in a micropipeline. Only its implementation is changed from globally synchronized using global timing assumptions to local handshake with none or local timing assumptions. This low-level sequential behavior implementation is done automatically and does not affect the design specification. Final implementation (and this is the main difference from RTL) will provide the result as soon as it can – not at the predetermined time as with synchronous RTL. It will signal the data availability and wait for the environment to acknowledge the data receipt to output the new result.

SADT flows differ in pipeline stage granularity. Inter-register handshake insertion approach where clock connected to registers is substituted by handshaking between the registers placed at the same points in the circuit (Fig. 1b) is used by NCL [19] and De-synchronization [20] flows.

The main distinctive feature of our approach [21] is that in addition to replacing global synchronization with local self-timed control we also remove functionally unnecessary synchronization and alter the granularity of pipelining (usually significantly decrease it down to the gate level Fig. 1c).

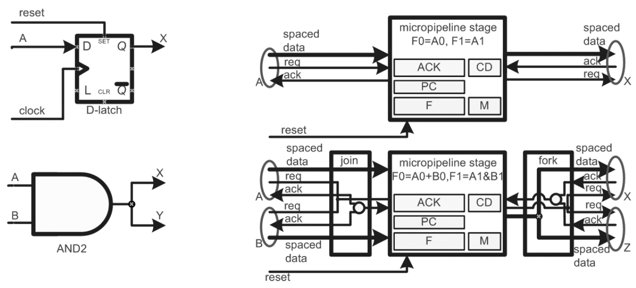
There are several reasons for gate-level pipelining: overcoming parameter variations, lower completion detection overhead (see section 3.2 for details on completion detection) etc. Particularly, we would like to mention that lower pipeline granularity is a way to improve performance. For security related applications gate level pipelining allows development of small power balanced gates (as explained in section 4) that can be used to automatically synthesize DPA resistant implementations.

The asynchronous mechanisms (including handshake communication) are hidden from the end circuit designer in the ***micropipeline cell library*** leaving the handshaking implementation to the library designer.

3.1 RTL to micropipeline re-implementation in our synthesis flow

Micropipeline synthesis (as a particular case of SADT methodology) consists of *three main stages: RTL synthesis*, *re-implementation and final mapping* explained as follows.

RTL implementation consists in synthesizing a synchronous implementation from HDL specification provided by the designer by a standard RTL synthesis tool. The only difference from standard RTL synthesis is that virtual library (imaginary) cells are used for synthesis. This step determines the implementation architecture. It can be tuned the same way it would be for RTL synthesis to trade-off area, performance and dynamic power consumption.



(a) Virtual library cell (b) Micropipeline cell in general

Fig. 2. Micropipeline synthesis examples: clocked latch (top) and AND2 gate with a fork (bottom).

Re-implementation takes the RTL netlist obtained in the previous step. First, *RTL functionality is identified*. Every combinational gate or a clocked latch (g_i) is represented with a library cell (see examples in Fig. 2). Clocked flip-flops are considered as pairs of sequentially connected latches (master g_{mi} and slave g_{si}) with alternative clock and are represented as two cells each. Every data wire (any wire except for clock and reset) is mapped to a cell connection. This way no additional data dependencies are added and no existing data dependencies are removed. Initial state of state holding gates (D-latches and D-flip-flops) is guaranteed by appropriate reset.

The algorithm is substitution based linear complexity assuming that for every virtual library cell there exists a **micropipeline library** cell or a previously synthesized module implementing functionality represented by the cell. This assumption is satisfied by targeting RTL synthesis to the virtual library that is functionally equivalent to the micropipeline library and by bottom-up synthesis of hierarchical designs

Next deadlock freedom is ensured and the **micropipeline netlist** is optimized using *slack matching* [22, 23] and other optimizations.

Fig. 2 presents identification and micropipeline synthesis examples for a clocked latch with fan-out of 1 and an AND2 gate with fan-out 2. The latch (Fig. 2a top) is connected to reset with its preset pin meaning that it is initialized to '1' in RTL implementation. During micropipeline synthesis an identity stage is chosen from the library that is initialized to dual-rail value of logical '1'. The combinational gate labeled with function A&B is implemented by a micropipeline stage with equivalent dual-rail functionality. The gate output depends on both inputs therefore the inputs must be synchronized by a *join* module. Likewise the output is split to X and Y what makes it necessary to synchronize the feedback acknowledgements with a *fork* module.

The nets not identified as special nets are treated as channels. Fig. 2 shows the general case of channel expansion using request (*req*), acknowledgement (*ack*) and dual-rail binary data wires. The join and fork module implementations are protocol dependent.

We have proved that asynchronous fine-grain pipelined circuit generated by our flow is live, safe and *flow-equivalent* to original specification (we borrow the notion of flow-equivalence and a method of proof from [20]). Flow-equivalence means that for each stage that corresponds to a latch in RTL implementation, the value stored at the i-th pulse of the control signal is the same as the value stored at the i-th cycle of the synchronous circuit.

3.2 Micropipeline stages

Numerous protocols and implementation styles have been developed for asynchronous micropipelines. The protocols fall into two groups [12]: *bundled data* using delay element to match the delay of data propagation through combinational logic and *completion detection* based. The latter encode data to include a *spacer* (no data value) in addition to logical '1' and logical '0' (e.g. like in dual-rail domino with data values "01" and "10" and the reset state "00"). Such an encoding along with monotonic transitions makes it possible to distinguish data from reset state by looking only at the data itself.

Data/spacer detection is called *completion detection*. For the above data encoding it can be implemented with a NOR gate per data channel. For multiple channels synchronization of single channel completions is implemented by a latch with the function $g = x1 \cdot x2 + g \cdot (x1 + x2)$, known as a Muller's C-element [15] shown on Fig. 3 as a circle with "C" inside. With no global synchronization a stage determines the time to precharge/evaluate by observing the feedback from data consumers. It can precharge when all consumers evaluated and evaluate when all of them precharged.

An example of dynamic implementation of a micropipeline stage cell implementing the AND2 function is shown on the Fig. 3. (This particular example illustrates the Reduced Stack Precharge Half-Buffer (RSPCHB) template from [24]. Note that RSPCHB is not balanced. It was not targeted to secure applications.) Block implementing the stage logical function is F. The rest of blocks are typical for most of the stages. *LReq* and *LAck* are left and *RReq* and *RAck* are right request and acknowledgement, ACK – handshake implementation, PC – phase (precharge/evaluate) control, CD –completion detection and M stands for memory. '*Staticizers*' (or *keepers*) formed by adding weak inverters as shown in Fig. 3, store the stage output value for an unlimited time eliminating timing assumptions. At the same time keepers solve the charge sharing problem and improve the noise margin of precharge style implementations. The *req* line is used in some protocols to signal data availability to the following stages while

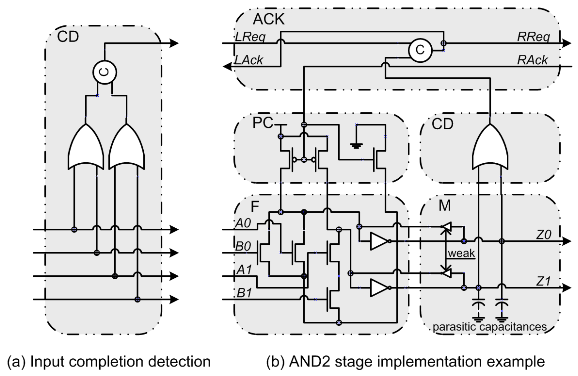

CD
ACK
LReq
LAck
RReq
RAck
C
PC
CD
F
M
A0
B0
A1
B1
Z0
Z1
weak
parasitic capacitances
(a) Input completion detection
(b) AND2 stage implementation example

Fig. 3. AND2 micropipeline stage dynamic implementation example.

the *ack* – to indicate that the data portion has been consumed. Depending on the communication protocol, some or all of the handshake events can be transmitted over the data lines so *req* and/or *ack* lines may not be needed.

A *dedicated micropipeline library* with each cell representing an entire micropipeline stage localizes in-stage timing assumptions and power balancing inside the cell thereby leaving it to the library designer. With delay-insensitive inter-stage communication the implementation functionality no longer depends on place & route. Note (Fig. 3) that memory and logic function implementation are of the same cost and speed as synchronous dual-rail domino counterparts. The main sources of area overhead are the Muller C-element for handshake control implementation (ACK), completion detection circuitry (CD) and ack/req synchronization (can be seen in Fig. 2).

3.3 Design flow and EDA support

Our synthesis flow consists of a reimplementation engine and a set of scripts responsible for implementing the user interface (commands) and interaction with the RTL synthesis tool. The engine incorporates VHDL and Synopsys Liberty parsers/generators to interface the design and library specifications with industrial tools. The RTL synthesis tool currently used in the flow is Synopsys Design Compiler®. This set of tools is targeted at micropipeline synthesis but it also automates some library installation tasks.

Library installation is executed once per library or every time the library is modified. This step is essential for the flow flexibility to use variety of micropipeline libraries. The flexibility is achieved through abstracting from particular micropipeline style(s) by defining a stage-cell as a pre-designed module implementing one or more functions of its inputs. Every data input or output is considered as a channel consisting of encoded data and zero or more handshake lines. On the example on the Fig. 2b a channel consists of dual-rail data, request and acknowledgement lines.

Virtual library is an imaginary single-rail synchronous RTL library functionally equivalent to the micropipeline library. The virtual library is generated from the micropipeline library during its installation. Cell AND2 on the Fig. 2a is a virtual library cell generated for stage AND2 implementation (Fig. 2b) found in micropipeline library. Area and delay characteristics of the virtual library cells are mapped from the corresponding micropipeline library cells to make optimization during the RTL synthesis meaningful.

4 Cell customization and security benefits

The previously described synthesis approach based on fine-grained templates is in large part independent of the detailed implementation of the template cell. Unlike other balanced asynchronous implementations and flows [14] which are much more restrictive in the structures which can be used, this general template is much more flexible and adaptable. Libraries optimized for balance, performance, power, or overhead can all be incorporated to meet the security and other design goals. Since the basic templates are based on differential dynamic cells almost all of the existing or novel dynamic circuit structures can be easily incorporated into an asynchronous standard-cell library. New circuit structures do not have to be redesigned or invented for a particular application in order to be incorporated into the flow, thus allowing reuse of intellectual property and further decreasing development time and time-to-market of complex designs.

For example, SABL gates [2, 3] can be easily adapted to the cells preserving all of their balance properties and enhancing their fault resistance and robustness. Addition of asynchronous control removes the clocking and timing difficulties normally associated with the gates and enhances their security applications due to the benefits of asynchronous behavior as mentioned in sec. 2. The function and operation of the additional asynchronous wrapper is almost completely data independent and only the completion detection of wrapper requires a trivial power balancing consideration which can be easily met with two additional minimal size transistors [9]. By simply using an unmodified SABL gate as the functional block of the asynchronous template and using the handshake circuitry of the template (like that presented in section 3.2 and shown on the Fig. 3) for the generation of the clock signal for the SABL gate as shown on the Fig. 4 a fully QDI balanced gate results. The resulting gate has identical balance to that of the original SABL gate.

Additionally, the explicit synchronization and completion detection of the asynchronous template allows for fewer restrictions on the design of the balanced functional block. Restrictions such as elimination of early propagation effect [25] which need to be explicitly considered in synchronous implementations are automatically satisfied. Explicit input completion can be incorporated to the design which coupled with the C-element will prevent evaluation until all of the input data has arrived and is ready.

Furthermore, the timing and voltage tolerance of the QDI implementation allows for more aggressive dynamic designs which can achieve better balance than

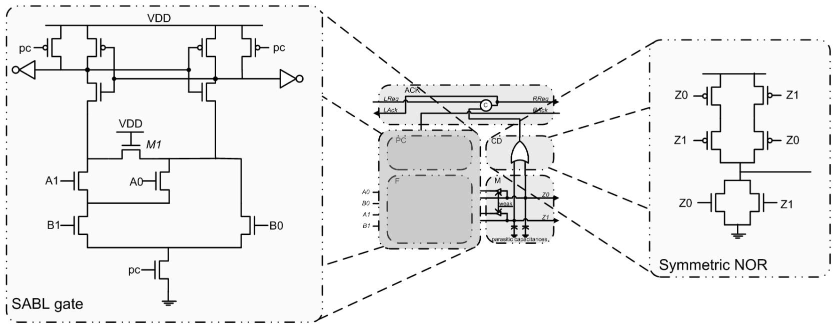


Fig. 4. Incorporation of a SABL gate into the QDI template.

previous designs. A balanced library designed specifically for the fine-grained asynchronous template called Balanced Symmetric with Discharge Tree (BSDT) gates was fully incorporated into the flow. The gates showed approximately 6 times better balance than the synchronous SABL implementations (Fig. 5) [9].

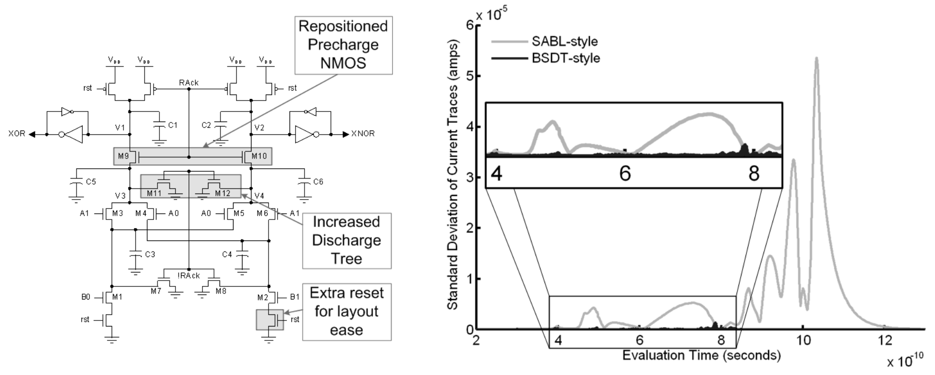


Fig. 5. BSDT-style XOR and the Standard Deviation of the evaluation phase of SABL and BSDT implementations.

Current versions of the balanced library cells based on existing balanced dynamic functional blocks still require balanced routing considerations. However, due to gate level asynchronous QDI nature of the method the resulting implementations are very tolerant of process/voltage variations. The natural tolerance of the template can allow more aggressive dynamic balancing techniques which can allow for routing independent gate design. We are currently developing a balanced library design which does not require balanced routing considerations.

In addition to allowing a more robust design for dynamic balanced function blocks the asynchronous handshake protocol and template adds natural fault resistance to the design. For the balanced asynchronous gates presented in [9] out of all the possible transistor level single stuck-at faults inside and outside of the complete asynchronous gate not a single fault changes the Boolean function of the gate. Almost 80% of the faults result in a pipeline stall which naturally prevents further data processing and creates deadlock within the pipeline (Tab. 1). That is the faults prevent or stop the necessary four phase handshake protocol between each gate thereby stalling the communication between dependent downstream gates and preventing any further data processing. To resolve the deadlock the pipeline requires an explicit reset which will clear all intermediate faulty data values inside the pipeline removing the possible source of fault attack information. Synchronous based balanced dynamic logic gates have no comparable property. This additional property should make it much harder to use invasive or semi invasive attacks on a circuit since almost all of the tampering would be detected by a pipeline stall. Additional error detection based on other high level fault-tolerant methods (i.e. error-detecting codes) [26] can be added easily due to the HDL synthesis support. Only a modified HDL specification incorporating fault-tolerance needs to be generated.

Table 1. Effects of stuck-at faults in asynchronous dual-rail gates

	Pipeline stall	No Logical effect on function	Created an alarm state
Buffer	75%	21%	4%
AND	73%	16%	10%
XOR	73%	16%	11%

We are currently performing a full analysis of the side-channel information leakage from sample implementations. Initial simulated power analysis attacks on the Sbox of the Data Encryption Standard (DES) indicate that the beneficial properties of the balanced dynamic gates and asynchronous circuits translate to the proposed implementations. The DPA was applied similarly to the attack performed on our previous balanced library implementation [8] and shows similar simulation results.

Since the design is based on components of previously evaluated methods and designs (i.e., QDI asynchronous design, dynamic balanced gates) it is expected that the good properties of the individual components should be preserved as indicated by the results of initial DPA simulations. Therefore with respect to power, fault and EMI channels the methodology is expected to be as secure, by construction, as the individual components prior to integration. We are currently evaluating the details and possible weaknesses resulting from the combination of the countermeasures but up to this point none have been found.

5 AES implementations comparison

To estimate efficiency of our flow [27] we compare performance of automatically synthesized synchronous and asynchronous balanced and unbalanced fine-grain pipelined implementations using our simple dynamic logic based micropipeline libraries using TSMC 0.18μm technology (obtained through MOSIS). One of the libraries – BSDT is a power balanced library implemented with minimum transistor sizes. Another – MPCHB (modified PCHB from [24]) optimized for performance.

The same RTL Electronic Code Book mode (unfolded 10-round) HDL specification of the AES has been used for all implementations. Synchronous RTL implementation was synthesized with the Artisan Sage-X^{TM} [28] standard cell library using the same (TSMC 0.18μm) technology. The non-pipelined implementation shows performance of 16MHz. Automatically pipelined (with Synopsys Design Compiler® "pipeline_design –period 0" command – maximum performance setting) synchronous implementation – performed at 45MHz.

Our asynchronous fine-graine pipelined implementations exceeds 35Gbps (298MHz*128bit where 128 bits is the input and cipher text word length) for balanced and over 62Gbps (482MHz*128bit) for unbalanced implementations.

Compare these performance numbers with commercial ASIC implementations like one from [29] available on the market today (25Gbps the word length of 256 bits – that scaled down to 12.5Gbps for the word length of 128 bits) or the best known academic custom (manual) design (546MHz) [30]. Note that in both cases there is no side-channel attacks protection. High performance and protection level results cost significant area overhead – the area of protected gate-level pipelined implementation approaches 30mm^2. Thanks to resistance to variation inherent to asynchronous micropipelines the implementation can operate at lower voltage with lower speed and lower power consumption.

Finally, we would like to note that both the MPCHB and BSDT libraries are under development and in the current stage feature logic gates (stages) up to 2 data inputs as well as the identity function (to be used for initialization and slack matching) micropipeline stages along with synchronization cells and a minimal set of standard logic cells. Design characteristics can be improved with better optimized and richer micropipeline library.

6 Conclusions and future tasks

The lack of industrial quality electronic design automation flow has limited the use of the most promising side-channel resistant circuit techniques: dynamic style balanced gates and asynchronous circuits. We have implemented a design methodology based on dynamic asynchronous micropipelines which allows full industrial quality EDA support without requiring additional training in asynchronous design. Moreover the methodology allows easy incorporation of existing synchronous dynamic gate designs and circuit structures. The combination of asynchronous operation and balanced dynamic gates allows automated standard-cell library based design highly resistant to side-channel attacks.

We recently discovered a new Combined Differential Power Analysis/Fault Injection (DPA/FI) attacks (or power attacks on faulty hardware) [31]. Our experiments indicate that this attack is potentially extremely dangerous since even Differential Power Analysis resistant (power balanced) implementations are vulnerable to DPA/FI attacks. No previous countermeasures have been specifically considered against this type of attacks. However, methodology based on asynchronous fine-grain pipelined power-balanced library is the approach which could provide for a high level of resistance against these new attacks.

Acknowledgements

This work was partially funded by Omnibase Logic Inc.

References

1. Fips pub 197: Advanced encryption standard, http://csrc.nist.gov.
2. Kris Tiri, Moonmoon Akmal, and Ingrid Verbauwhede. A dynamic and differential cmos logic with signal independent power consumption to withstand differential power analysis on smart cards. In *28th European Solid-State Circuits Conference (ESSCIRC 2002)*, 2002.
3. Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. *Design Automation and Test in Europe Conference (DATE 2004)*, 2004.
4. David Chinnery and Kurt Keutzer. Closing the Gap between ASIC & Custom. Tools and Techniques for Gigh-Performance ASIC Design. Kluwer Academic Publishers, 2002.
5. David Harris. Skew-Tolerant Circuit Design. Morgan Kaufmann Publishers, 2001.
6. Kris Tiri, Wei Hwang, Alireza Hodjat, Lai Bo-Cheng, Yang Shenglin, P. Schaumont, and I. Verbauwhede. Prototype IC with WDDL and differential routing - DPA sesistance assessment. In *Chyptographic Hardware and Embedded Systems - CHES*, pages 354–365, Edinburgh, 2005. LNCS3659, Springer.
7. J. McCardle and D. Chester. Measuring an asynchronous processor's power and noise. In *SNUG*, 2001.
8. Konrad J. Kulikowski, Ming Su, Alexander Smirnov, Alexander Taubin, Mark G. Karpovsky, and Daniel MacDonald. Delay insensitive encoding and power analysis: A balancing act. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 116–125, 2005.
9. Daniel Jay MacDonald. A Balanced-Power Domino-Style Standard Cell Library for Fine-Grain Asynchronous Pipelined Design to Resist Differential Power Analysis Attacks. Master of Science Thesis, Boston University, 2005.
10. H. Li, A. Markettos, and S. W. Moore. Security evaluation against electromagnetic analysis at design time. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2005.
11. A.Taubin, K. Fant, and J. McCardle. Design of delay-insensitive three dimension pipeline array multiplier for image processing. *ICCD*, 2002.
12. Jens Sparsø and Steve Furber, editors. Principles of Asynchronous Circuit Design: A Systems Perspective. Kluwer Academic Publishers, 2001.
13. Alain J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C. A. R. Hoare, editor, *Developments in Concurrency and Communication*, UT Year of Programming Series, pages 1–64. Addison-Wesley, 1990.

14. G. F. Bouesse, M. Renaudin, S. Dumont, and F.Germain. DPA on quasi delay insensitive asynchronous circuits: Formalization and improvement. In *DATE*, 2005.
15. David E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching*, pages 204–243. Harvard University Press, April 1959.
16. M. Renaudin, P. Vivet, and F. Robin. A design framework for asynchronous/ synchronous circuits based on CHP to HDL translation. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 135–144, April 1999.
17. Catherine G. Wong and Alain J. Martin. High-level synthesis of asynchronous systems by data-driven decomposition. In *Proc. ACM/IEEE Design Automation Conference*, pages 508–513, June 2003.
18. Ivan E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, June 1989.
19. Michiel Ligthart, Karl Fant, Ross Smith, Alexander Taubin, and Alex Kondratyev. Asynchronous design using commercial HDL synthesis tools. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 114–125. IEEE Computer Society Press, April 2000.
20. J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou. De-synchronization: synthesis of asynchronous circuits from synchronous specifications. *IEEE Transactions on Computer-Aided Design.* (To appear).
21. A. Smirnov, A. Taubin, and M. Karpovsky. An automated fine-grain pipelining using domino style asynchronous library. In *ACSD 2005: Fifth International Conference on Application of Concurrency to System Design*, St.Malo, France, 2005. IEEE CS Press.
22. Peter A. Beerel, Mike Davies, Andrew Lines, and Nam-Hoon Kim. Slack matching asynchronous designs. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 184–194, March 2006.
23. Piyush Prakash and Alain J. Martin. Slack matching quasi delay-insensitive circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 195–204, March 2006.
24. Recep O. Ozdag and Peter A. Beerel. High-speed QDI asynchronous pipelines. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 13–22, April 2002.
25. K. Kulikowski, M. Karpovsky, and A. Taubin. Power attacks on secure hardware based on early propagation of data. In *12th IEEE International On-Line Testing Symposium*, 2006.
26. K. Kulikowski, M. Karpovsky, and A. Taubin. Robust codes for fault attack resistant cryptographic hardware. In *Fault Diagnosis and Tolerance in Cryptography, 2nd International Workshop*, pages 1–12, Edinburgh, 2005.
27. Weaver: GTL synthesis flow. http://async.bu.edu/weaver/.
28. TSMC 0.18μm process 1.8-volt Sage-X standard cell library databook, September 2003.
29. High performance AES cores for ASIC - http://www.heliontech.com, 2005.
30. A. Hodjat and I. Verbauwhede. Area-throughput trade-offs for fully pipelined 30 to 70 Gbits/s AES processors. *IEEE Transactions on Computers*, 55(4), 2006.
31. K. Kulikowski, M. Karpovsky, and A. Taubin. DPA on faulty cryptographic hardware and countermeasures. In *Fault Diagnosis and Tolerance in Cryptography, 3nd International Workshop*, 2006.