Synthesis of Secure FPGA Implementations

Kris Tiri and Ingrid Verbauwhede

UCLA Electrical Engineering Department,
7440B Boelter Hall, P.O. Box 951594, Los Angeles, CA 90095-1594
`{tiri, ingrid}@ee.ucla.edu`

Abstract. This paper describes the synthesis of Dynamic Differential Logic to increase the resistance of FPGA implementations against Differential Power Analysis. The synthesis procedure is developed and a detailed description is given of how EDA tools should be used appropriately to implement a secure digital design flow. Compared with an existing technique to implement Dynamic Differential Logic on FPGA, the technique saves a factor 2 in slice utilization. Experimental results also indicate that a secure version of the AES encryption algorithm can now be implemented with a mere 50% increase in time delay and 90% increase in slice utilization when compared with a normal non-secure single ended implementation.

1 Introduction

For embedded applications, which typically have a short product life span, time to market is crucial. Field Programmable Gate Arrays (FPGA's) are essential components to obtain a short design time. Compared to a full custom ASIC design, they are cost efficient, easier to manage and can immediately be put into operation. Furthermore, they can continuously be reprogrammed during and after the design. This feature allows for upgrades, which will maximize the product life span and enhance the reliability. Yet FPGA's have an edge over software solutions on embedded processors as they have lower power consumption and a higher data throughput, which are both important design criteria for battery-powered devices.

Consumers attach great value to security related issues such as privacy and prevention of unauthorized use. Hence most present-day embedded devices have cryptographic capabilities. Depending on the application, these features are used for authentication, data integrity and/or confidentiality. System designers spend much design effort in developing secure protocols and selecting a strong encryption algorithm to achieve the security level envisioned in the specifications.

Any security application however, is only as safe as its weakest link. Information related with the physical implementation of the device, such as time delay and power consumption, has been used repeatedly to find the secret key in so-called Side Channel Attacks [1]. Especially the Differential Power Analysis (DPA) [2] is of great concern as it is very effective in finding the secret key and can be mounted quickly with off-the-shelf devices. The attack is based on the fact that logic operations have power characteristics that depend on the input data. It relies on statistical analysis to retrieve the information from the power consumption that is correlated to the secret key.

Any concrete implementation, including the FPGA, carries Side Channel Information. Hence SCA's have been identified as an important open issue related to the general security of cryptographic applications on FPGA's [3]. Much effort has already gone into setting up the DPA for FPGA's. Just recently, several research groups have independently presented the idea to attack FPGA implementations with DPA [4],[5],[6].

It is expected that general countermeasures, which have been developed in the past on the architecture level or the algorithmic level, can also be implemented on an FPGA. To some extent, they will help in concealing the supply current variations. Yet none has ever proven really successful and effective in thwarting the DPA. It is our opinion that the solution is at the logic level. Indeed, the fact that the power consumption of a single logic gate, which is the most elementary building stone of the complete

encryption module, is controlled by both the logic value and the sequence of its input signals forms the basis of DPA.
We have previously presented a logic level design methodology to implement a secure DPA resistant crypto processor on FPGA [7]. The methodology can easily be integrated in a common automated FPGA design flow. The technique combines standard FPGA building blocks to make secure compound standard cells, which have a reduced power signature. The compound cells mimic custom designed standard cells that have been used to thwart DPA [8]. In this manuscript, we study the synthesis aspects in order to reduce the area consumption and time delay.
The next section briefly introduces Wave Dynamic Differential Logic (WDDL). WDDL is the cornerstone of the logic level design methodology for a secure DPA proof design. Section 3 discusses a technique to combine several compound WDDL gates into 1 slice. This reduces the area consumption and time delay. Section 4 describes the clustering procedure of the synthesis methodology. In section 5, a secure digital design flow is presented together with detailed instructions in order to implement the design flow with available EDA tools. In section 6, an experiment is setup in which the performance of the methodology is evaluated. Finally a conclusion will be formulated.
For this work, we have used the Virtex-II FPGA as our implementation platform [9]. The concepts behind the synthesis methodology however, are universal and can be applied on any FPGA product.

2 Wave Dynamic Differential Logic

The power consumption of traditional standard cells and logic is dependent on the signal activity, both the transitions and the sequence. This is the fundamental reason information is leaked through the power supply and DPA attacks are possible. To address this problem we have introduced a family of secure compound standard cells, referred to as Wave Dynamic Differential Logic [7]. A WDDL gate uses a fixed amount of charge for every transition and has for that reason at all times a constant power consumption independent of the signal transitions. It has the extra advantage that it does not use custom designed standard cells or advanced circuit techniques and is as a result it is also applicable to FPGA.
WDDL is a Dynamic and Differential Logic (DDL). A Differential Logic family uses the true and the false representation of the input signals to generate the true and false representation of the output signal. A Dynamic Logic family alternates precharge and evaluation phases, in which both outputs are precharged to ‘1’ and exactly 1 of the 2 outputs is evaluated to ‘0’ respectively. Consequently, DDL charges in every cycle a capacitance.
The basic building block of a Virtex-II FPGA is known as a slice and consists of two 4-input, 1-output look up tables (LUT’s), some multiplexers and registers [9]. A compound WDDL standard cell occupies 1 slice of which both LUT’s are used, 1 for each output. Fig. 1 shows the LUT definition and allocation of the secure WDDL AND- and OR-gate implementation. Letters ‘G’ and ‘F’ stand for the G-LUT and F-LUT respectively, which are the denominations for the LUT’s of the Virtex-II FPGA.

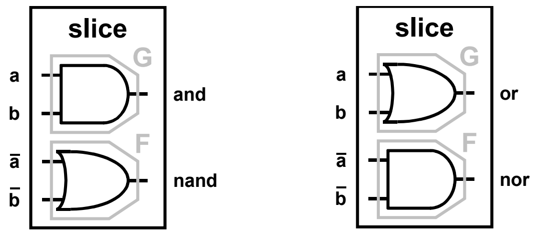


Fig. 1. LUT definition and allocation of secure WDDL gates: AND-gate (left); and OR-gate (right)

The set of secure compound logic gates is restricted to the WDDL AND- and OR-gates. Since any logic function in Boolean algebra can be implemented with the and-, or- and invert-operator and given that the compound gates have differential outputs, this library is sufficient to implement any digital design. The library size has been restricted in order to assure that every compound standard cell of an arbitrary logic function has exactly 1 output transition per cycle [7]. Only if each gate has a switching factor of 100%, it is possible to have an input independent power consumption.
The precharge phase of this logic is unique. A module in WDDL precharges without distributing the precharge signal to each individual gate as is normally done in a dynamic logic style. As shown in Fig. 1, each WDDL gate consists of a parallel combination of a single ended and-gate and a single ended or-gate. Whenever the inputs to these gates are pre-'dis-charged to '0' the output will pre-'dis'-charge to '0' as well. During the precharge phase, the input vector of the design is set at all '0's and the module calculates the result. Each individual gate will eventually have all its inputs at '0'; evaluate its output to '0' and pass this '0' value to the next gate. One could say that the precharge signal travels over the FPGA as a '0'-wave, hence Wave DDL.
Besides that each compound standard cell has exactly 1 output transition per cycle, it is essential that the standard cell always charges ideally the same load capacitance. The load capacitance consists out of the interconnect capacitances of the wires that run between 2 gates and out of the intrinsic capacitances of the in- and output pins of the gates. Both LUT's actually have the same structure and the capacitances at the corresponding differential input signals are alike just as the capacitances at the output pins [10]. There will however be a difference in the interconnect capacitance due to routing variations between the two output signals. In order to minimize this effect the 2 LUT's, which make up a compound standard cell, are placed adjacent and in the same slice as in Fig. 1. By doing this the differential signals need to travel the same distance to the gate they connect to.
Experimental results have shown that WDDL is an efficient technique to achieve a reduced power signature of FPGA implementations [7]. By way of illustration, Fig. 2 shows a measured output voltage transient for 10 clock cycles of a balanced XOR-tree test circuit. Measurements are performed with a HP 54542C oscilloscope. The prototyping board is the Xilinx Virtex-II Development Kit by Avnet Design Services [11]. The figure shows that the non secure single ended design suffers from glitches, which are the spurious signal transitions. Even partial glitches have a significant contribution in the dynamic power dissipation. The WDDL implementation on the other hand, has as expected only one transition. Whenever the output out does not switch, the differential output $\overline{\text{out}}$ switches.

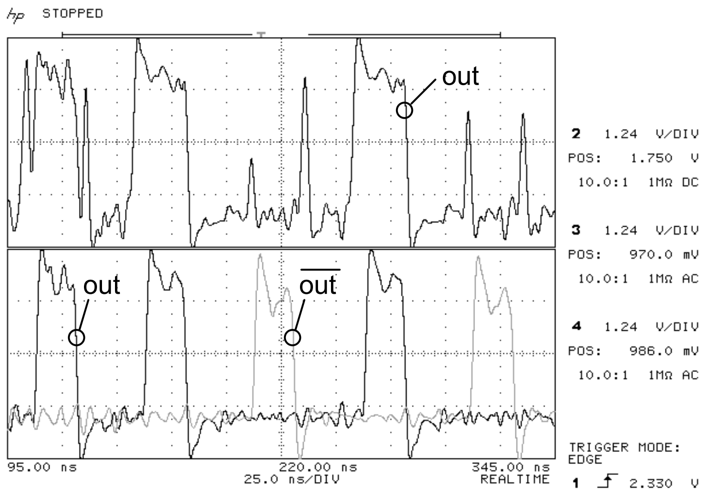


Fig. 2. Measurement of output transient: single ended design (top); and WDDL implementation (bottom)

3 Slice Compaction

Currently, 2 LUT's are used to build a compound WDDL gate. Each LUT is needed to generate one of the differential outputs. It is however, possible to add more functionality, or in other words more logic gates, into each LUT. A restricted combination of compound logic gates will result in a new compound logic gate with the same Dynamic Differential behavior. This practice will reduce the area and timing requirements.
Fig. 3 and Fig. 4 show an example. Fig. 3 (left) depicts the single ended logic function to be implemented. The WDDL implementation that results from our original methodology is shown in Fig. 3 (right). In this design each gate is replaced by its corresponding WDDL gate. In total, 6 slices are occupied. The logic depth is 3.

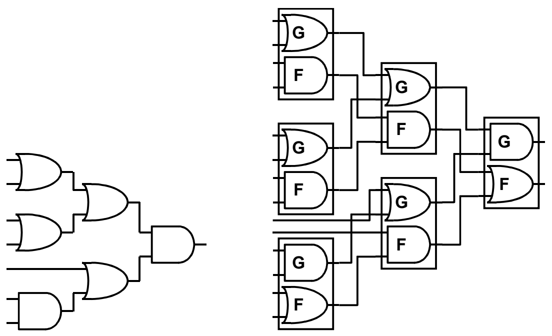


Fig. 3. WDDL design procedure: original single ended logic function (left); and DPA proof implementation (right)

For transparency of the figure, Fig. 3 (right) is repeated in Fig. 4 (left). Yet the F-LUT's have been colored light gray. This reveals that the implementation can actually be divided into two distinct components, which contain the G-LUT's and F-LUT's respectively.
The two parts are also dual. One component can be derived from the other by inverting the inputs and by redefining each LUT from the and- and or operator to the or- and and-operator respectively. The G-LUT's calculate the true output, the F-LUT's the false output. Furthermore, since the original module is implemented in WDDL, only 1 output will have a switching event in any given clock cycle. As a result, the collection of G-LUT's and the collection of F-LUT's can be considered as 1 large G-LUT and 1 large F-LUT respectively. Or in other words, the entire module behaves as one WDDL gate.
This observation is accurate for a module of any given size, including submodules. A cluster formed by an arbitrary collection of G-LUT's and the cluster formed by the corresponding F-LUT's behave as a compound WDDL-gate and can in fact be implemented as 1 compound WDDL-gate in the 2 adjacent LUT's of a slice. An LUT on the Virtex-II platform has 4 inputs and 1 output. Here, the clusters can have at most 4 inputs and 1 output.
Fig. 4 (right) shows the implementation after clustering. This implementation occupies only 2 slices and has a logical depth of 2. Recall that the original WDDL implementation requires 6 slices and has a logical depth of 3.

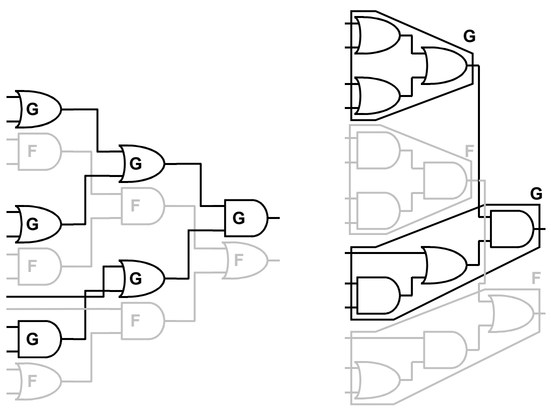


Fig. 4. Cmpaction through clustering: original WDDL implementation (left); and clustered WDDL implementation (right)

The WDDL combinatorial logic should be compacted according to the clustering-idea presented in Fig. 4. The rest of this work describes the design flow to perform synthesis for an FPGA implementation of DPA-proof combinatorial logic.

4 Logic Synthesis

The kernel of DPA-proof logic synthesis is a clustering algorithm. Given a DPA-proof implementation consisting solely out of secure compound AND- and OR-gates, the algorithm partitions the implementation in groups of LUT's with 4 or less distinct inputs and 1 output. Each group will form together with their corresponding dual group a secure compound gate and will be mapped onto adjacent LUT's within the same slice.

4.1 Clustering Challenges

A group of LUT's can be divided into many partitions. Certain partitions are more advantageous in area consumption and time delay than others. In order to partition the gates appropriately, several issues should be considered.
A small area is the primary design criterion for most implementations. The area consumption is proportional to the total number of LUT's, or in other words the number of groups in the partitioning. For high speed designs on the other hand, a small time delay is critical. The time delay is proportional to the critical path, which is defined by a combination of the logical depth and the fanout of the LUT's. The fanout of an LUT, or in other words a group in the partitioning, is the number of LUT's that are driven by the output of the LUT.
An exhaustive search is guaranteed to find the global optimum. Yet this will be very computational intensive for a medium to large design. A heuristic search is a more appropriate approach. In this case, the degree of compaction will be influenced by the traversal order of the gates. Clusters that have been defined in the initial phase, affect the clustering in the subsequent phase. Fig. 5, which for clarity only shows the G-LUT's, depicts 2 partitions. Fig. 5 (right), where the clustering is carried out from the output backwards to the inputs, requires twice as much slices then the optimal clustering depicted at the left.

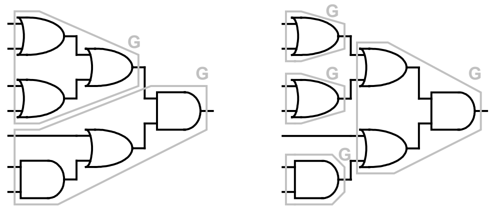


Fig. 5. Influence of initial cluster on degree of compaction (only G-LUT's shown): left-to-right traversal (left); and right-to-left traversal (right)

The introduction of redundancy may also reduce the total number of clusters. Fig. 6, which for clarity only shows the G-LUT's, provides an example. The straightforward clustering, which is depicted in Fig. 6 (left) requires 3 slices. On the other hand, when the upper OR-gate is duplicated, only 2 slices are necessary. This is shown at the right.

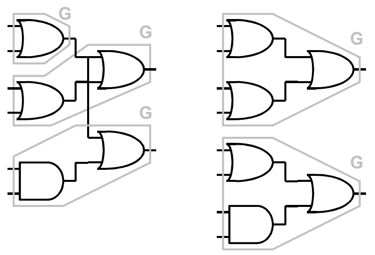


Fig. 6. Increased compaction through introduction of redundancy (only G-LUT's shown): without redundancy (left); and with redundancy (right)

The following subsection describes an efficient and simple procedure to perform the clustering.

4.2 Clustering Procedure

Fig. 4 (right), which depicts the optimized DPA-proof implementation, could also have been obtained through a simple transformation of the synthesized single ended FPGA implementation. As can be seen in Fig. 7, the clustered implementation is a parallel combination of the synthesized single ended FPGA implementation and the dual of this single ended implementation. Between the optimized single ended implementation and the DPA-proof implementation there is now an overhead of a factor of 2 in the area consumption and if wire delays remain unaffected, no overhead in the time delay.
This means that we can use existing synthesizing tools for FPGA's to cope with the clustering and its challenges. These EDA tools have been developed for years and are being improved continuously. The tools have for instance appropriate algorithms, cost functions and stopping criteria for heuristic searches and will introduce redundancy. The design can be optimized for many specific applications such as minimum area or maximum speed.

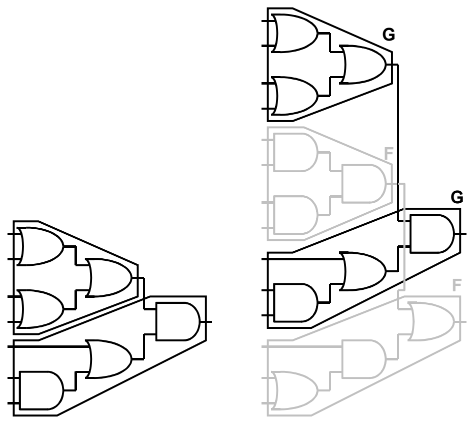


Fig. 7. Clustering through transformation: synthesized single ended design (left); and WDDL implementation as parallel combination of synthesized single ended design and its dual

Yet to implement an arbitrary logic function, several inversions may be necessary. An issue associated with these inversions prohibits a straightforward implementation of this technique.
This is best seen with an example. Fig. 8 (A) shows a single ended design implemented with the basic library of and-, or-, and inverter-gates. The synthesized single ended FPGA implementation is shown in Fig. 8 (B). Note that inside 1 LUT, there is an inversion. This is not a good partitioning. The precharge '0' at the input of the inverter is propagated as a '1' and consequently at least 1 of the 2 dual LUT's will have a '1' at the output during the precharge phase. Hence, the precharge wave is halted.
The WDDL implementation obtained through to the original design methodology is shown in Fig. 8 (C). Here the inverters have been removed. Instead the outputs of the secure compound gate that precedes the inverter have been exchanged. Now, there is no inverter present anymore to halt the precharge wave. This procedure however, interconnects the G- and F-LUT's. The far right and-gate, implemented in the G-LUT, is driven by both a G-LUT and an F-LUT. This would imply that it is necessary to take both G- and F-LUT's into consideration during the clustering and that only steppingstone C is a valid approach.

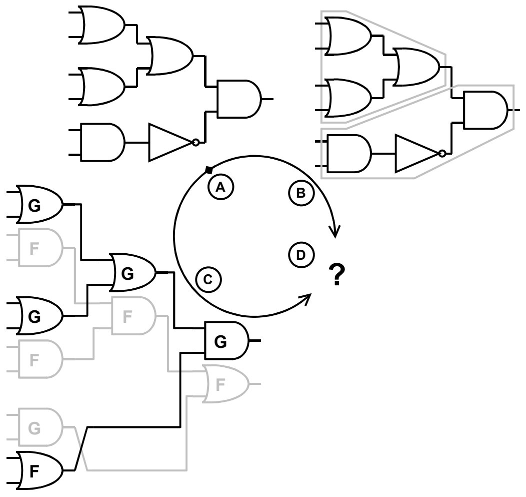


Fig. 8. Inversion mixes G-LUT's and F-LUT's: arbitrary logic function with inversion (A); synthesized single ended design (B); and original WDDL implementation (C)

Fortunately, a small change in the design methodology allows using steppingstone B to find the compacted version of steppingstone C. We propose the methodology depicted in Fig. 9 to synthesize secure FPGA implementations. There are 4 main stages:

1. The combinatorial logic is synthesized with a limited standard cell library. The library contains the and-, or- and invert-operator. This step is in common with the original DPA-proof design methodology.
2. The intermediate design is stripped off the inverter-gates. The input of each inverter becomes a global output; the output of each inverter becomes a global input.
3. The intermediate design without inversions is synthesized for FPGA implementation.
4. The design is doubled. Each LUT of the intermediate design is implemented in a G-LUT, the dual of the LUT is implemented in the adjacent F-LUT. Finally, the in- and outputs resulting from stripping the inverters in stage 2 are reconnected. The inversions are established by switching the differential connections.

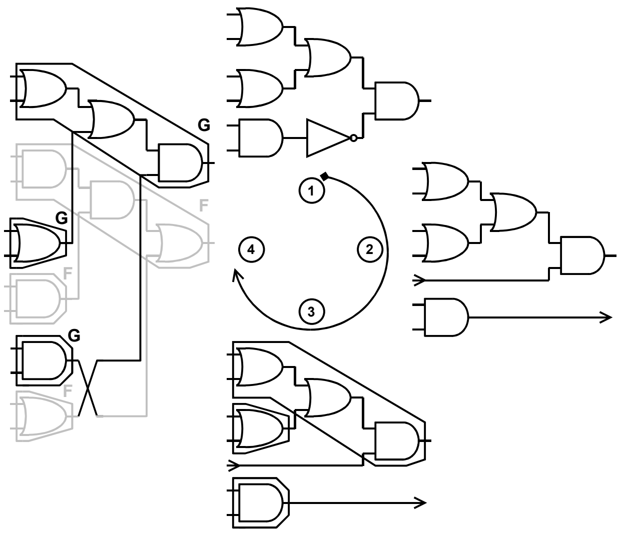


Fig. 9. Steppingstones for synthesis of DPA-proof combinatorial logic for FPGA implementations

5 Secure Digital Design Flow: Detailed Description

Fig. 10 shows the adapted secure digital design flow that implements secure DPA resistant FPGA logic with the synthesis procedure derived in the previous sections. In synchronous logic, the logic design of a module can be done with a standard hardware description language, such as Verilog or VHDL. Synthesis is done with a subset of the standard cell library. Subsequently, a script, e.g. in PERL or AWK, parses the netlist and removes the inverters. This netlist is synthesized by a synthesize tool for FPGA implementations. The resulting netlist is parsed. Each LUT is doubled. This means that the dual of each LUT will be placed in the adjacent LUT and is connected with the inverted signals. The inversions that have been stripped out previously are reestablished by switching the appropriate lines. Finally, this netlist is read in by the synthesize tool for FPGA implementations to Place & Route the design. At the end, the programming file can be generated.

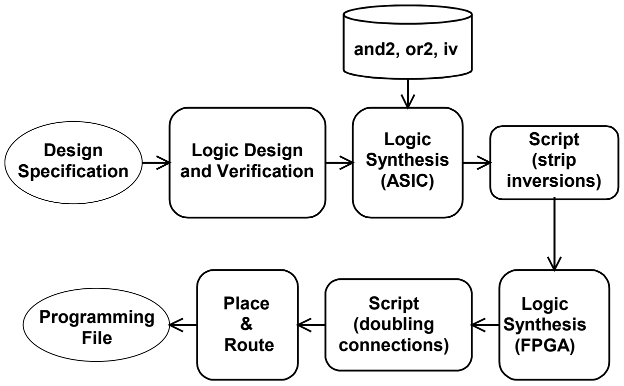
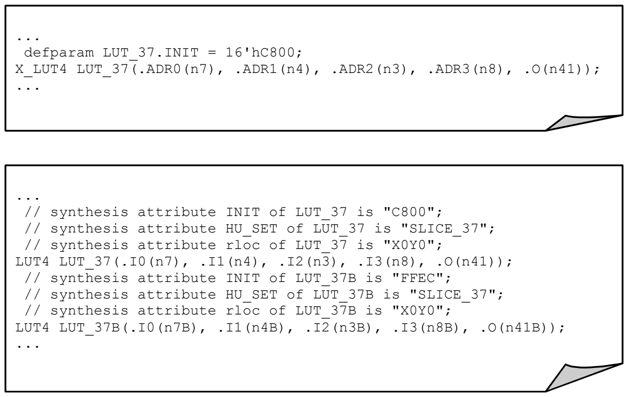


Fig. 10. Adapted secure digital design flow for FPGA's.

We will now present a detailed description of how certain EDA tools are used appropriately for our specific design objectives. Only the steps that were not obvious or simple are discussed. Note that we do not advocate the use of any specific tool; many other tools could have been used.
We have used DesignAnalyzer from Synopsys for step 1. In this tool, it is possible to restrict the library by setting the set_dont_use variable for all gates of the library except the 3 gates.
We have used the XST Verilog design flow of ISE of Xilinx to synthesize the FPGA implementation in step 3. The translate step generates a Verilog netlist that can easily be parsed. This netlist consists out of declarations of primitive modules of the device. Fig. 11 shows how the declarations are transformed from single ended LUT's to WDDL gates. The first expression defines the behavior of the LUT. To find the dual expression for the F-LUT, each bit of the binary representation is inverted and the order of the bits is reversed. The second expression defines the hierarchical subset of the LUT. The subset name should be unique for every WDDL gate. Finally in the third expression, the LUT's are placed in adjacent LUT's of a slice. This is done by setting the relative location statement of both LUT's, which belong to the same hierarchical subset, to X0Y0.



```
...
 defparam LUT_37.INIT = 16'hC800;
X_LUT4 LUT_37(.ADR0(n7), .ADR1(n4), .ADR2(n3), .ADR3(n8), .O(n41));
...
```

```
...
 // synthesis attribute INIT of LUT_37 is "C800";
 // synthesis attribute HU_SET of LUT_37 is "SLICE_37";
 // synthesis attribute rloc of LUT_37 is "X0Y0";
LUT4 LUT_37(.I0(n7), .I1(n4), .I2(n3), .I3(n8), .O(n41));
 // synthesis attribute INIT of LUT_37B is "FFEC";
 // synthesis attribute HU_SET of LUT_37B is "SLICE_37";
 // synthesis attribute rloc of LUT_37B is "X0Y0";
LUT4 LUT_37B(.I0(n7B), .I1(n4B), .I2(n3B), .I3(n8B), .O(n41B));
...
```

Fig. 11. Netlist transformation: original single ended LUT declaration (top); and declaration of WDDL gate (bottom)

6 Experimental Results

Many block ciphers are based on the sequential application of confusion and diffusion. Confusion is typically provided by some form of substitution and is executed by so-called substitution boxes. These operations are often the main components of an encryption algorithm and are responsible for the largest share of the area consumption and time delay. We implemented each substitution-box of 3 prevalent encryption algorithms, which are the Kasumi encryption algorithm [12], the DES encryption algorithm [13] and the AES encryption algorithm. [14]. Kasumi is the encryption algorithm in 3G cellular standard. DES and AES have originally been developed to protect sensitive data of the US federal government and are now used in a broad range of applications.
For each substitution box, 3 designs have been implemented: (1) "original WDDL", which is the DPA-proof implementation resulting from the original design methodology; (2) "Differential synthesis", which is the result of a straightforward synthesis of the differential netlist of the original WDDL implementation; and (3) "synthesized WDDL", which is the result of a compaction of the original WDDL logic according to the synthesis methodology discussed in this work.
The difference between the original WDDL implementation and the Differential synthesis implementation is that the latter has only behavioral information of the module. The synthesis tool only knows what are the in- and outputs and how to calculate the outputs from the inputs. The tool is free to implement this, or in other words to map the functionality onto the LUT's, how it wants as long as the outputs are correct. For the original WDDL implementation on the other hand, we have done the synthesis step and the mapping step ourselves. Strictly speaking, the tool is only involved with the Place & Route step.
The Differential synthesis implementation has been implemented to have a benchmark circuit. Note that this implementation does not have Dynamic Differential behavior and is certainly not DPA proof. Many of the Differential synthesis implementations require an odd number of LUT's. Therefore they can never have the same number of switching events in every cycle.
The programming files have been generated for a xc2v1000 device of the Virtex2 family with a bg575 package and speed grade -6. Synthesis steps 1 and 3 and Place & Route have been done with the default settings of the different tools. This means that the designs are optimized for speed and with normal effort. Place & Route has been done with the same pin locations for each implementation of an encryption algorithm.
For each implementation, we calculated the slice utilization, which denotes the number of slices occupied by the design and the delay of the critical path. The timing report of the Place & Route tool only gives an estimate of the delay between the input and output pins. The numbers that we present have been adjusted for the intrinsic delay of the input and output buffer. These delays are subtracted, as they are independent of the delay of the modules. The delay numbers include the influence of the interconnect wireloads. They have been obtained after Place & Route.
Fig. 12 shows the slice utilization for the 8 Sboxes of the DES algorithm. There is an almost constant reduction of a factor 2 between the original WDDL design and the synthesized design. It is quite remarkable that there is also an important difference between the benchmark implementation and the compacted WDDL implementation. The Differential synthesis implementation requires up to 25% more slices then the synthesized WDDL implementation.
We suppose that, even though the Differential synthesis implementation has not the stringent constraints to generate Dynamic and Differential logic, it performs worse because the compacted WDDL has been implemented with the knowledge that this is a dual module and that the in and output signals are differential signals.

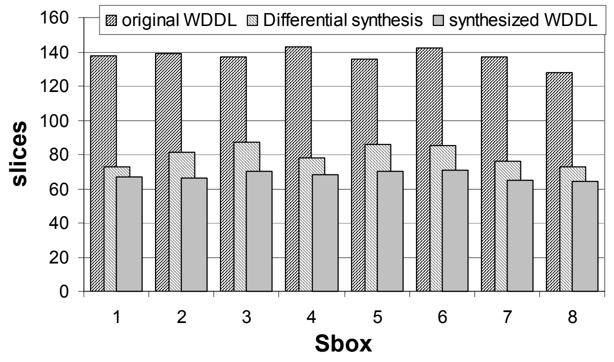


Fig. 12. DES algorithm: slice utilization of substitution boxes

Table 1 presents the slice utilization for the AES and the Kasumi algorithm. The reduction factors are 2.4 and 1.7 for the AES algorithm and the Kasumi algorithm respectively. For the Kasumi algorithm, the results of the Differential synthesis are as good and better than the ones after compaction. The Kasumi algorithm incorporates many xor operations. Since these operations are inverting operations, it is hard to generate large clusters.

Table 1. AES and Kasumi algorithm: slice utilization of substitution boxes

	original WDDL	Differential synthesis	synthesized WDDL
AES-Sbox	**797**	**357**	**327**
Kasumi-S7	**249**	**142**	**142**
Kasumi-S9	**303**	**157**	**171**

Security never comes for free. We have also implemented a single ended version of the modules. There is an increase of a factor 5 for the DES and the Kasumi algorithm The DPA proof AES implementation however, only requires 1.9 times the slices of the single ended implementation.

Fig. 13 and Table 2 present the critical path delays. There is a reduction in time delay between the original WDDL implementation and the synthesized delay. Yet the improvements are not that remarkable as for the area decrease. The delay numbers seem to be closely related to the logical depth of the module. The logical depth has only decreased with 1 or 2 levels. For the DES algorithm the synthesized WDDL performs better then the benchmark implementation. This is not the case for the 2 other encryption algorithms.

Compared to the single ended implementation, there is an increase of about a factor of 2 in time delay. The AES algorithm performs again better. The DPA proof implementation has only a 50% penalty in the time delay.

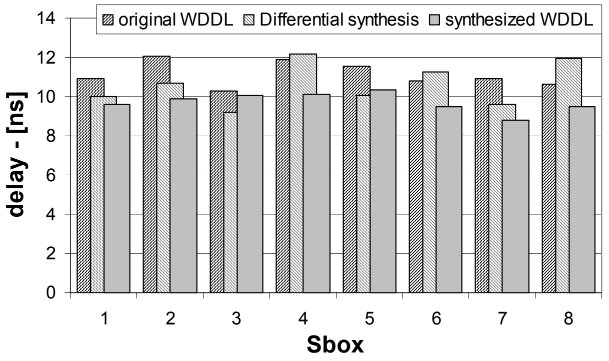


Fig. 13. DES algorithm: critical path delay of substitution boxes

Table 2. AES and Kasumi algorithm: critical path delay of substitution boxes

	original WDDL	Differential synthesis	synthesized WDDL
AES-Sbox	17.95	13.66	14.98
Kasumi-S7	15.31	11.24	11.57
Kasumi-S9	14.46	9.96	11.25

In our experiments, the synthesis tool, which is used to find the clusters, used the default settings and minimized the time delay. Yet, some timing information is unavailable to the tool. The in- and outputs, which have been created by stripping the inverters from the design, are only virtual in- and outputs for the purpose of our synthesis goals. The synthesis tool does not know this. The tool will optimize each path delay such that they are smaller then the user constraints. Yet eventually after the doubling the LUT's and reestablishing the connections, the delays are summed and may be larger than the original critical path the synthesis tool was optimizing for. Therefore, it seems better to optimize for area.

7 Conclusions

We have presented a design methodology to synthesize secure DPA resistant logic for FPGA implementations. The methodology has been completely described and an exact implementation procedure has been provided.
Experimental results have shown that compared with the original WDDL, the slice compaction offers a reduction of at least a factor of 2 in slice utilization. The gain in time delay depends largely on the algorithm and can go up to 30%. The methodology seems perfectly applicable for the AES algorithm. Compared with a single ended design, the overhead in time delay and slice utilization is restricted to a factor 1.5 and 1.9 respectively.
Experiments have also shown that the synthesis methodology achieves smaller utilization factor and time delay than a professional synthesize tool.

Acknowledgements

The authors would like to acknowledge the support of National Science Foundation (CCR-0098361).

8 References

1. E. Hess, N. Janssen, B. Meyer and T. Schuetze. “Information Leakage Attacks Against Smart Card Implementations of Cryptographic Algorithms and Countermeasures – a Survey,” Proceedings of Eurosmart Security Conference pp. 55–64, June 2000.
2. P. Kocher, J. Jaffe and B. Jun, "Differential Power Analysis," Proceedings of Advances in Cryptology (CRYPTO 1999), Lecture Notes in Computer Science Volume 1666, pp. 388–397, Jan. 1999.
3. T. Wollinger and C. Paar, “How Secure Are FPGAs in Cryptographic Applications?” Proceedings of International Conference on Field Programmable Logic and Applications (FPL 2003), Lecture Notes in Computer Science Volume 2778, pp. 91-100, Sept. 2003.
4. B. Örs, E. Oswald and B. Preneel, “Power-Analysis Attacks on an FPGA--First Experimental Results,” Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003), Lecture Notes in Computer Science Volume 2779, pp. 35–50, Sept. 2003.
5. F. Standaert, L. van Oldeneel to Oldenzeel, D. Samyde and J. Quisquater, “Differential Power Analysis of FPGAs: How Practical is the Attack?,” Proc. International Conference on Field Programmable Logic and Applications (FPL 2003), Lecture Notes in Computer Science Volume 2778, pp. 701–711, Sept. 2003.
6. L. McDaniel, “An Investigation of Differential Power Analysis Attacks on FPGA-based Encryption Systems” M.S. thesis, Virginia Polytechnic Institute and State University, May 2003. (http://scholar.lib.vt.edu/theses/available/etd-06062003-163826/unrestricted/Larry_McDaniel.pdf)
7. K. Tiri and I. Verbauwhede, “A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation” in Proc. of Design Automation and Test in Europe Conference (DATE 2004), pp. 246-251, Feb. 2004.
8. K. Tiri and I. Verbauwhede, “Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology,” Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003), Lecture Notes in Computer Science Volume 2779, pp. 125–136, Sept. 2003.
9. Xilinx “Virtex-II Platform FPGAs: Detailed Description,” 2002. (http://www.xilinx.com/bvdocs/publications/ds031-2.pdf)
10. L. Shang, A. S. Kaviani, and K. Bathala, “Dynamic Power Consumption in Virtex-II FPGA Family,” Proceedings of the 2002 ACM/SIGDA 10th International Symposium on Field-Programmable Gate Arrays, pages 157–164. ACM Press, 2002.
11. Avnet Design Services: Xilinx Virtex-II Development Kit. http://www.ads.avnet.com PN: ADS-XLX-V2-DEV1500
12. Universal Mobile Telecommunications System (UMTS); “Specification of the 3GPP confidentiality and integrity algorithms; Document 2: Kasumi algorithm specification,” (3GPP TS 35.202 version 5.0.0 Release 5), June 2002. (http://pda.etsi.org/pda/)
13. NIST Federal Information Processing Standards (FIPS) PUB 46-2 “Data Encryption Standard,” Dec. 1993. (http://www.itl.nist.gov/fipspubs/fip46-2.htm)
14. NIST Federal Information Processing Standards (FIPS) PUB 197 “Advanced Encryption Standard,” Nov. 2001. (http://csrc.nist.gov/CryptoToolkit/aes/)