The Dissertation Committee for Yu-Wei Lee Certifies that this is the approved version of the following dissertation:

TEST AND SECURITY IN A SYSTEM-ON-CHIP ENVIORNMENT

Committee:

Nur Touba, Supervisor

David Z. Pan

Earl E. Swartzlander Jr

Lizy K. John

Gang Huang

TEST AND SECURITY IN A SYSTEM-ON-CHIP ENVIORNMENT

by

Yu-Wei Lee, B.S.; M.S.;

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2017

Dedication

To my parents, my family and my dearest wife.

Acknowledgements

I would like to thank Prof. Nur Touba, for being my academic supervisor at Graduate school in University of Texas, Austin. There are simply too many things I have learned from him during this journey. Most importantly, I learned how not to settle and always do things the right way. These experiences will not only help me throughout my career, but will also be valuable to every decision I will be making in life.

I thank Prof. Lizy John, Prof. David Pan and Prof. Earl Swartzlander for agreeing to be on my Ph.D. committee and reviewing this dissertation.

I am also grateful to Dr. Gang Huang for being on my committee, providing his in-depth advices from the industry perspective.

I would not be where I am now without the support of my family. I want to thank my parents for their support to everything I do. If I am going to have kids, I hope I can be as supportive as you two were to me. I want to thank my sister for her pep talks throughout the years. I hope to remain a good brother you love to hang out with, as well as a cool uncle to your adorable kids.

I am the luckiest man on earth to have my wife with me all the time. She is my ultimate cheerleader and I hope am doing the same for her. This have been an incredible journey, and I cannot wait hear the next movement of our symphony.

Test and Security in a System-On-Chip Environment

Yu-Wei Lee, Ph.D.

The University of Texas at Austin, 2017

Supervisor: Nur A. Touba

This dissertation outlines new approaches for test and security in a System-on-Chip (SoC) environment. A methodology is proposed for designing a single test access mechanism (TAM) architecture on each die with a "bandwidth adapter" that allows it to be efficiently used for multiple different test data bandwidths in three-dimensional integrated circuits (3D-IC) using through-silicon vias (TSVs). In this way, a single test architecture can be re-used for pre-bond, partial stack, and post-bond testing while minimizing test time across all phases of test. Unlike previous approaches, this methodology does not need multiple TAM architectures or reconfigurable wrappers in order to be efficient when the test data bandwidth changes. In industry, sequential linear decompression is widely used to reduce data and bandwidth requirements. A new scheme using a multiple polynomial linear feedback shift register (LFSR) with rotating polynomial is proposed here to increase encoding flexibility resulting in higher compression ratios. An algorithm is described to assign test cubes to polynomials in a way that enhances encoding efficiency. For hardware security, a new attack strategy against logic obfuscation is described here. It is based on applying brute force iteratively to each logic cone one at a time and is shown to significantly reduce the number of brute force key combinations that need to be tried by an attacker. It is shown that inserting key gates based on MUXes is an effective approach to

increase security against this type of attack. In data security for hardware, existing techniques for computing with encrypted operands are either prohibitively expense (e.g., fully homomorphic encryption) or only work for special cases (e.g., linear circuits). A lightweight scheme implemented at the gate-level is proposed for computing with noise-obfuscated data. By carefully selecting internal locations for noise cancellation in arbitrary logic circuits, the overhead can be greatly minimized. One important application of the proposed scheme is for protecting data inside a computing unit obtained from a third party IP provider where a hidden backdoor access mechanism or hardware Trojan could be maliciously inserted.

Table of Contents

List of Tables

List of Figures

1. Introduction

Improvements of IC technology increase the transistor density but also bring more challenges for design, verification, and test. As the scale and capabilities of ICs continue to grow, more effort is needed to ensure logic correctness and integrity. In design-for-test (DFT), scan based design greatly improves circuit testability. Scan chains are formed by connecting storage elements in a design into shift registers in test mode to provide better control and observability. The IEEE 1149.x family, also called Joint Test Action Group (JTAG) standard [IEEE 1149.1-2001], is one of the most widely adopted standards for DFT. Figure 1 presents an example of the JTAG architecture. Basic elements in JTAG include scan cells, test access ports (TAP), TAP controller, and instruction registers for the JTAG controller.

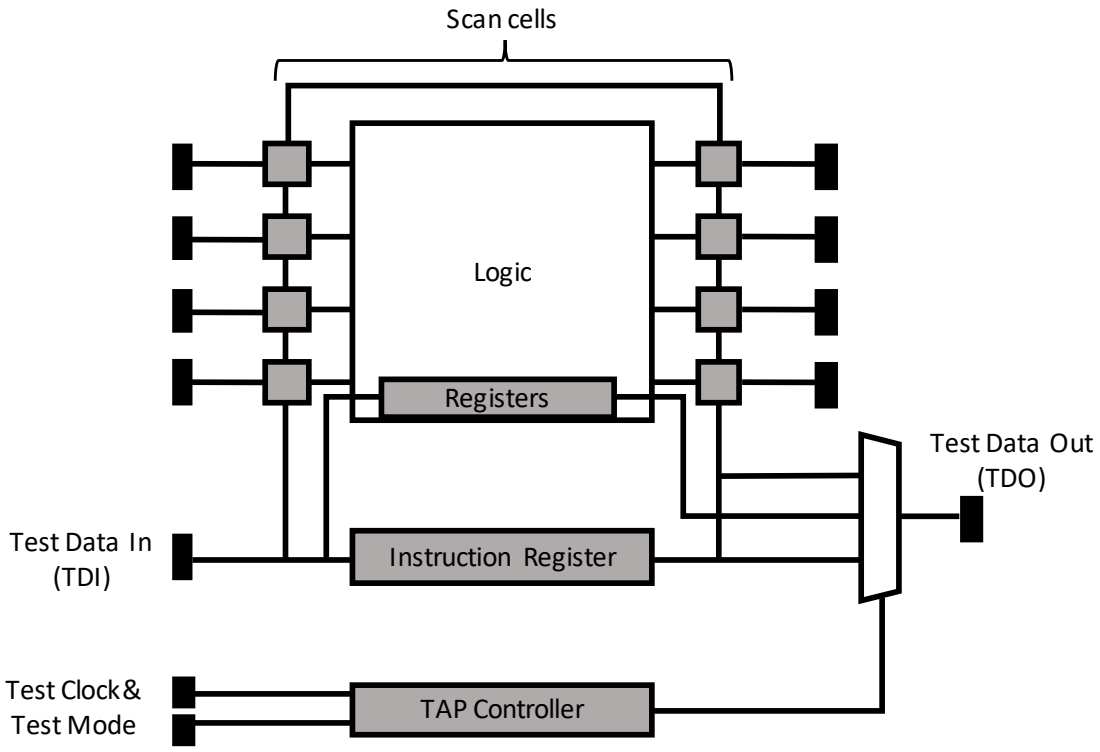


Figure 1. JTAG Architecture

A System-on-Chip (SoC) can be viewed as a collection of multiple cores implementing different applications. Each core can be separately designed and integrated later onto a single piece of silicon. SoCs allows intellectual property (IP) cores to be easily reused and shortens design time for complex applications. DFT for SoCs may be complicated by the fact that each IP core may have different specifications for test. The IEEE 1500 standard was proposed to address the SoC core based test problem [Marinissen 99]. It inherits most of the properties of IEEE 1149.1 with some additional hardware components such as a test wrapper for each core, a single source and sink for test access, and an on chip test-access mechanism (TAM) to connect wrappers to the test controller. Figure 2 shows an example of an IEEE 1500 system in a SoC consisting of *N* cores. Optimizing TAMs is often a challenging problem in DFT since each core in a SoC may have a different specification for test, and the TAM is responsible for control and distribution of test data for each core under test.

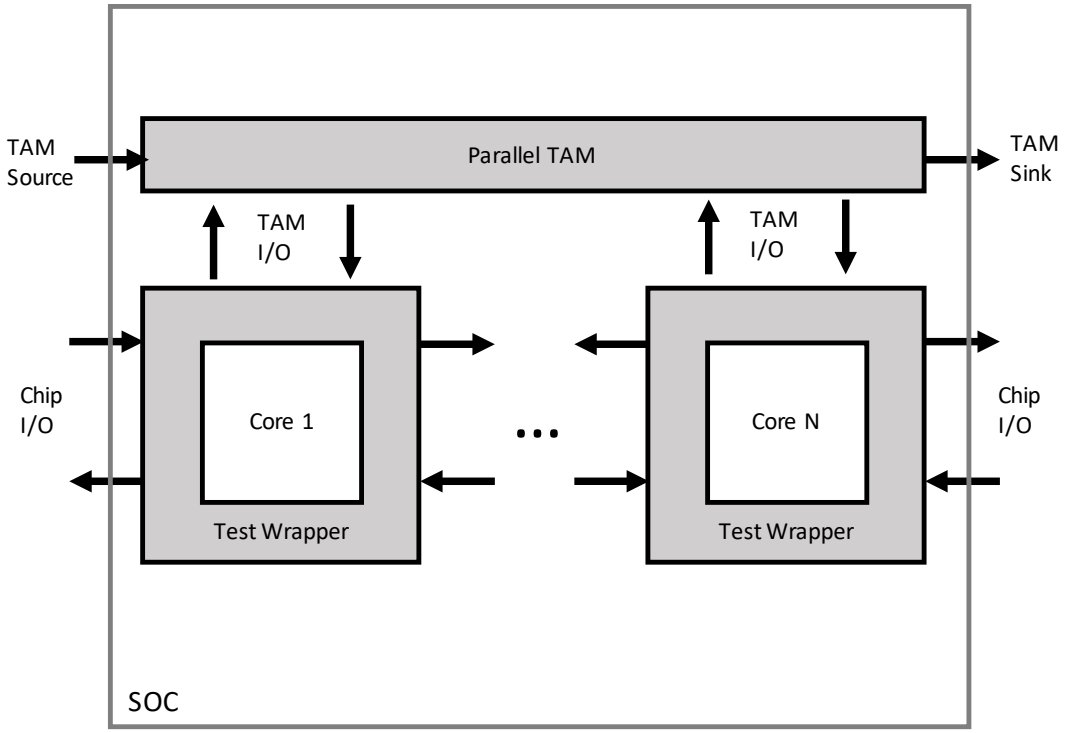


Figure 2. IEEE 1500 in a SoC of N Cores

Manufacturing SoCs implemented with three-dimensional integrated circuits (3D-ICs) have become an increasingly used approach that provides technological advantages over traditional 2D design. In a 3D-IC, dies are manufactured individually before stacking. Before stacking, pre-bond testing can be used to screen out defective die and only known-good die (KGD) are used when constructing the stack. Furthermore, during construction of the stack, additional known-good stack (KGS) testing can be done after each die is added. During each phase of test, the access mechanism and bandwidth requirement may be different. In chapter 2, a methodology is proposed for designing a single TAM architecture on each die with a "bandwidth adapter" that allows it to be efficiently used for multiple different test data bandwidths.

As SoC designs scale, more components are being integrated. Even with more components, test circuitry usually does not scale as quickly as the functional part. This creates challenges for test data compression because each component needs a separate testcubes, but still must sharing the same tester bandwidth. Test data compression is widely used to compress the amount of data stored on the tester. This helps to reduce tester storage requirements and improve test time as less data has to be transferred over the limited test data bandwidth between the tester and chip-under-test. Sequential linear decompression is a highly efficient technique used to reduce tester bandwidth requirement as well as data storage. This approach is based on a linear feedback shift register (LFSR) with multiple locations to inject variables from tester. Test cubes can be encoded by solving a system of linear equations where each equation corresponds to a care bit in the test vector being encoded [Könemann 01], [Krishna 01], [Rajski 04].

One common drawback for conventional LFSR decompression, however, is the encoding flexibility of the test cubes is restricted by the LFSR polynomial. Some test cubes simply are not encodable by some polynomials, but may be encodable by a different polynomial. Encoding flexibility can be greatly increased with a multiple-polynomial LFSR [Hellebrand 95] in which multiple polynomials can be selected to encode different test cubes. In chapter 3, a new data compression scheme based on a rotating multiple

polynomial LFSR is proposed to improve data compression in sequential linear decompression. The implementation eliminates the conventional control bits needed to switch the polynomials, and is made efficient in terms of hardware overhead through exploiting properties of primitive polynomials.

Advancement in DFT improves controllability and observability, but creates concerns that sensitive information about an IC might be exposed. The security concerns for an IC are applicable to both logic and data. IP piracy has become a problem for designers because an unauthorized party may learn and reproduce the design without consent by using reverse engineering. Another problem is that a malicious party may obtain enough knowledge about the design to insert additional circuitry to disrupt or change normal functionality. This kind of malicious circuitry is known as hardware Trojans. Logic obfuscation has been proposed to protect ICs against these attacks [Roy 08, 10], [Chakraborty 09a], [Baumgarten 10], [Rajendran 12, 14]. One approach is to use combinational obfuscation where "key gates" are inserted in a design which have primary inputs that are referred to as "key inputs". The design will only function correctly if the correct key values are used [Roy 08, 10]. In chapter 3, a new form of attack against logic obfuscation through analysis of logic cones is considered. In addition, a way to counter such an attack with MUXes as key gates is described.

In addition to protecting the hardware, the data stored and processed in the hardware also needs to be protected. This concern is becoming increasingly prominent as more ICs are used for processing critical and personal information. For encryption in software, a groundbreaking scheme called fully homomorphic encryption (FHE) is proposed in [Gentry 09]. FHE allows computing with encrypted data so that an attacker could not obtain the plain text even with the ability to snoop information within the computation hardware. FHE can be implemented in software with very high performance overhead and implementation cost in hardware is unrealistically high. Chapter 4 proposes a lightweight hardware computing scheme where the computation is conducted on obfuscated data. The idea for protecting data in the computing unit is to insert noise in the

input data and cancel it out both partially internally in the computing unit as well as fully at the output of the computing unit. A key idea in the proposed approach is to reduce the complexity of the noise cancellation logic by carefully selecting internal locations to do local noise cancelling. This is done in a way that prevents more than one input per gate from propagating noise thereby avoiding the complexity that arises from reconvergent noise propagation paths. While the proposed methodology does not provide the level of strong encryption that fully homomorphic encryption would provide, it has the advantage of being lightweight, easy to implement, and can be deployed with relatively minimal performance impact. One important application of this approach is for protecting data inside a computing unit obtained from a third party IP provider where a hidden backdoor access mechanism or hardware Trojan could be maliciously inserted. Chapter 5 concludes the dissertation with a summary of the contributions and suggestions for future work.

2. Unified 3D Test Architecture for Variable Test Data Bandwidth Across Pre-Bond, Partial Stack, and Post-Bond Test[1]

Three-dimensional integrated circuits (3D-IC) using through-silicon vias (TSVs) are an important new technology that provides a number of significant advantages including increased functional density, shorter interconnect, higher performance, and lower power. Stacking in a 3D-IC can be done wafer-to-wafer (W2W), die-to-wafer (D2W), or die-to-die (D2D). W2W allows higher manufacturing throughput, but achieving a good compound yield is difficult. In D2W and D2D, pre-bond testing can be used to screen out defective die and use only known-good die (KGD) when constructing the stack. Furthermore, during construction of the stack, additional known-good stack (KGS) testing can be done after each die is added to the stack which adds additional test cost, but has been shown to payoff in reducing overall costs by avoid additional processing in some cases [Taouil 10].

In order to do pre-bond testing on the non-bottom layers, it is necessary to add probe pads for test purposes. This is because the TSV tips as well as the microbumps are too small to be probed and are sensitive to scrub marks [Marinissen 09]. These probe pads are a test overhead that takes up a lot of space and limits the locations where TSVs can be placed which puts constraints on the design and floorplan of a die. Minimizing the probe pads is very important. In post-bond test, the bottom layer is accessed through the normal functional pins, and test data is transported to the upper layers via test elevator TSVs.

In 2D testing, test access mechanisms (TAMs) are used to deliver test data to core wrappers which interface with the scan chains in the cores. Procedures for optimizing the test architecture (i.e., selecting the width of TAMs, which cores are connected to each TAM, and optimizing the test wrappers) to minimize test time and satisfy power constraints have been well studied [Iyengar 02], [Goel 02], [Xu 05], [Larsson 05]. These algorithms

[1] The work in this chapter is published in [Lee 13]: Y.-W. Lee and N.A. Touba, "Unified 3D Test Architecture for Variable Test Data Bandwidth Across Pre-Bond, Partial Stack, and Post-Bond Test," *Proceedings of IEEE Symposium on Defect and Fault Tolerance*, Paper 7.5, 2013. Yu-Wei Lee is the author and Nur. A. Touba is the supervisor.

are based on having a fixed test data bandwidth available from the tester. Based on this bandwidth they optimize the test architecture to minimize test time.

For 3D testing, procedures for optimizing the test architecture for post-bond test under a constraint on the number of TSVs used as test elevators to bring test data from the bottom layer to the non-bottom layers has been proposed in [Wu 08] and [Noia 10a]. In [Noia 10b], an optimization procedure is described for generating a test schedule for each stage of KGS testing (as each die is added to the stack) followed by the final test.

In pre-bond test, the test data bandwidth available from the tester for non-bottom dies is severely constrained because of the need to minimize the number of probe pads and thus will likely be different from the test data bandwidth used in post-bond test. This creates a challenge for optimizing the test architecture to minimize the overall test time corresponding to the sum of the test time for pre-bond test of each die plus the time for post-bond test of the final stack. In [Jiang 09], a test optimization procedure is proposed for this problem, but it assumed that the test elevators to the non-bottom layers could be probed in pre-bond test, however this would require a large number of probe pads to accomplish. In [Jiang 12], the problem is addressed by designing two TAM architectures, one optimized for the test data bandwidth available for pre-bond test and the other optimized for the test data bandwidth available for post-bond test. The key idea is to try to share the test wires in pre-bond and post-bond test as much as possible to minimize overhead.

In this chapter, a methodology is proposed for designing a single TAM architecture on each die with a "bandwidth adapter" that allows it to be efficiently used for multiple different test data bandwidths. In this way, a single test architecture can be re-used for pre-bond, partial stack, and post-bond testing while minimizing test time across all phases of test. Unlike previous approaches, this methodology does not need multiple TAM architectures or reconfigurable wrappers in order to be efficient when the test data

bandwidth changes. This helps to simplify the test architecture and minimize routing and overhead costs. The work in this chapter is published in [Lee 13].

2.1. PROPOSED SCHEME

The idea in the proposed scheme is to optimize the test architecture on a particular layer for the maximum test data bandwidth, *n*, that the layer will receive during any phase of test (pre-bond, partial stack, or post-bond). Then for any phase of test where the bandwidth to the layer is less than *n*, a bandwidth adapter is used to handle the bandwidth mismatch.

An input bandwidth adapter is shown in Figure 3. It takes as an input k bits of test data each clock cycle where *k* is less than or equal to *n*. It stores the *k*-bits received each clock cycle in a *2n-1* bit buffer at the location pointed to by the *in_pointer* and then increments the pointer by *k* mod *2n-1*. When the difference between the *in_pointer* and *out_pointer* indicates that n or more bits are ready in the buffer, then the bandwidth adapter outputs the n bits pointed to by the *out_pointer* and increments the *out_pointer* by *n* mod *2n-1*. When the *n* bits are outputted, a pulse is generated on the *layer_test_clock* so that the internal layer TAM will transport the data. The bandwidth adapter can be designed in a general fashion to handle arbitrarily different test data bandwidths. Note that it is not necessary for *n* to be a multiple of *k*. Any value of *k* can be handled.

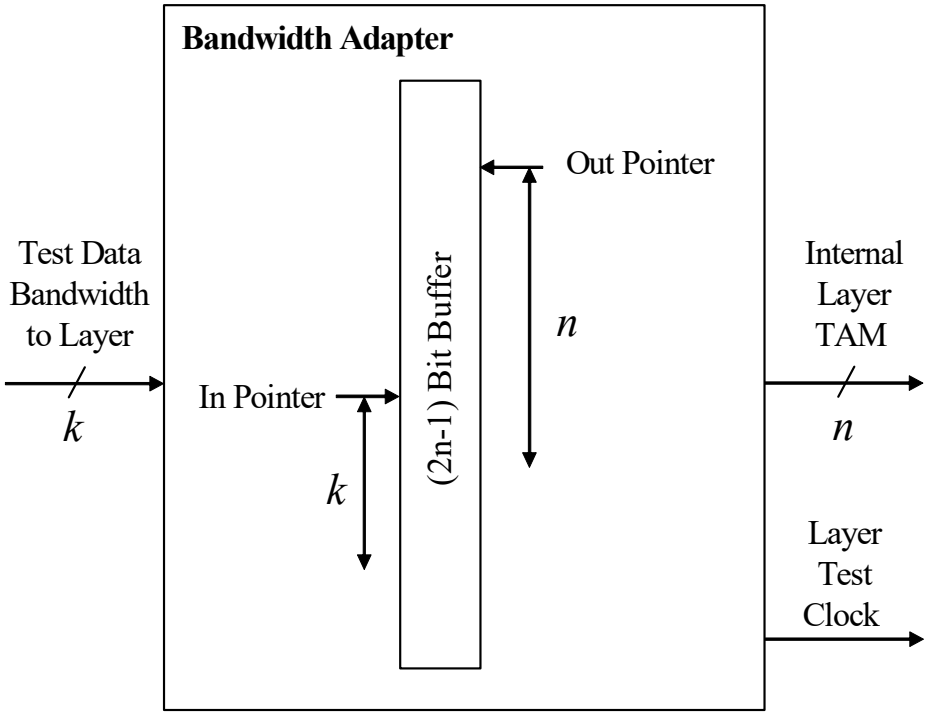


Figure 3. Input Bandwidth Adapter

By the same token, an output bandwidth adapter can be designed to handle the output response. It is the inverse of the input bandwidth adapater. It takes as an input the *n* bits of the internal layer TAM and the *layer_test_clock*, and it outputs *k* bit test data every clock cycle.

The proposed bandwidth adapter is totally independent to the underlying test architecture and thus is not limited to scan-based design. Note that some previous research has implemented the idea of variable bandwidth using reconfigurable wrappers which connect multiple scan chains to allow a smaller bandwidth to fill the scan chains [Khoche 01], [Koranne 03]. Serial/parallel conversion has also been used in [Noia 11] in the context of optimizing firm dies in post-bond test.

Figure 4 illustrates using a bandwidth adapter to unify the test architecture on a die for both pre-bond and post-bond test. The incoming bandwidth for pre-bond and post-bond tests are different because during pre-bond test, a smaller number of probe pads are used so the tester can directly probe the die, whereas during post-bond test, a larger number of test elevator TSVs are used to bring test data up from the pins in the bottom layer which are connected to the tester. The test architecture is optimized for the larger post-bond test data bandwidth, and then during pre-bond test, when the dies are tested separately, test vectors are brought in through the probe pads to the input bandwidth adapter which allows it to drive the larger number of TAM lines, and the output TAM lines go through an output bandwidth adapter to drive a smaller number of output probe pads.

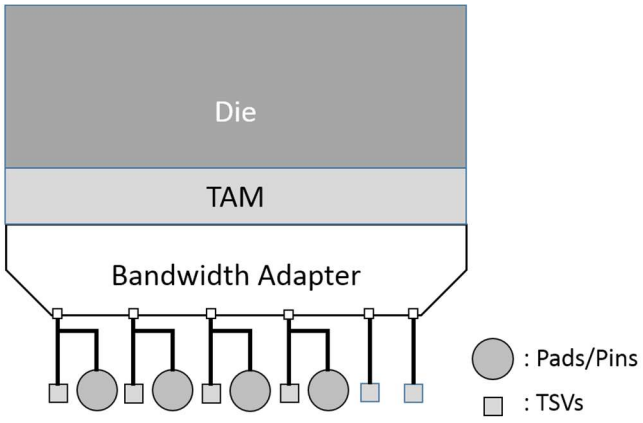


Figure 4. Block Diagram for Using Bandwidth Adapter

Figure 5 illustrates all the different phases of test using the bandwidth adapter. As mentioned previously, partial stack test may be optional. With the help of a bandwidth adapter, the insertion of a partial stack utilizes the fact that only a subset of dies from the final stack test is tested in this stage. With the bandwidth adapter, the underlying test architecture does not need to be redesigned for this additional test. That is, the partial stack test can be viewed as an intermediate stage of a post-bond test. The dies in these

intermediate stages can use any bandwidth different from the final stack test within the limitation on the number of test elevators and test pins.

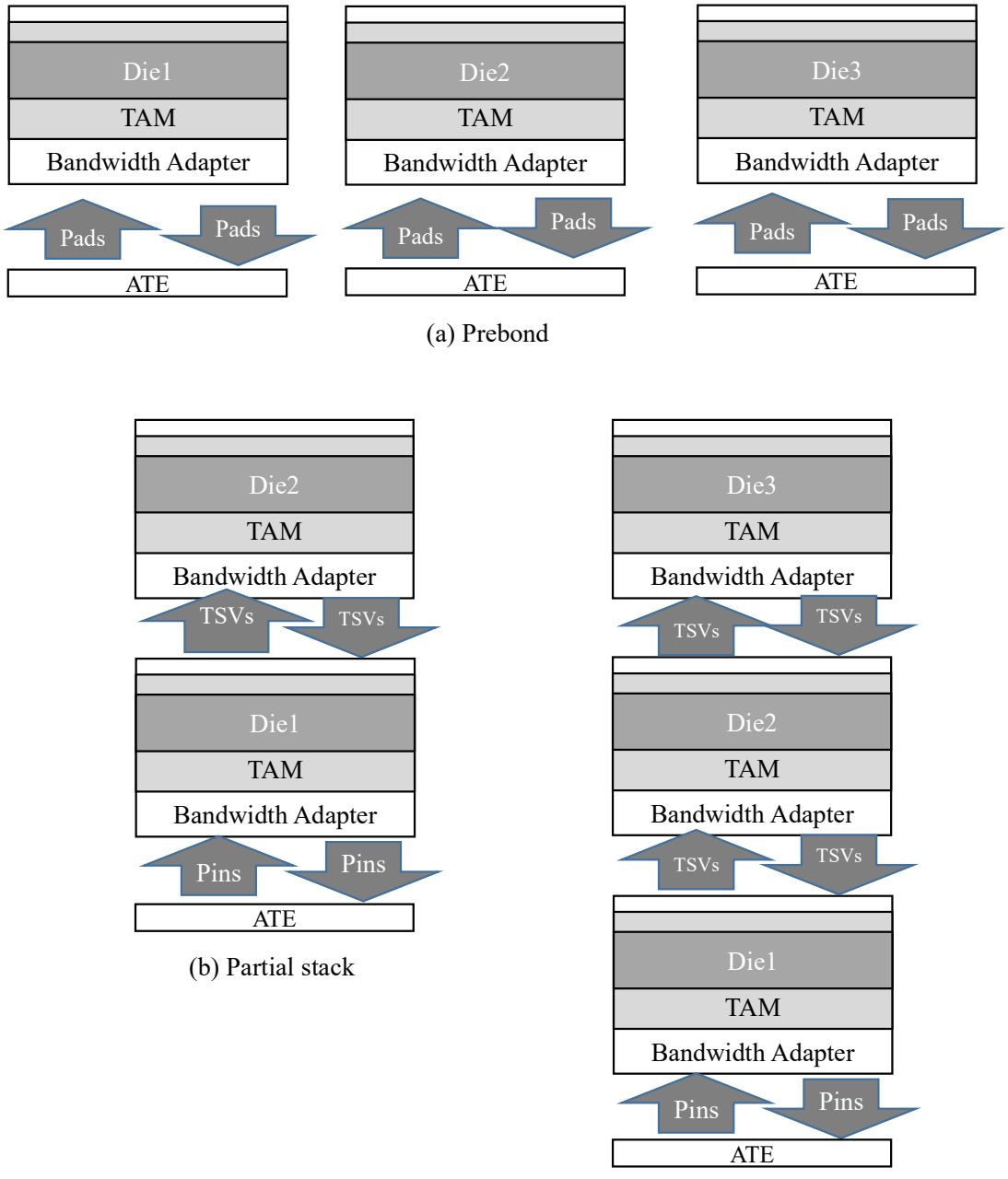


Figure 5. Unified TAM, Bandwidth Adapter and the Corresponding Test Flow

2.2. SELECTING BANDWIDTH FOR EACH STAGE OF TEST

This section describes how the test bandwidth is allocated at each stage of test to minimize the overall test time under the assumption of a unified test architecture with the proposed bandwidth adapter. There are two different scenarios considered in this section. The first scenario is only a pre-bond and post-bond test. The second scenario also includes partial stack tests, which can be viewed as a subset of the final stack test.

If only pre-bond and post-bond test are performed, the bandwidth used for pre-bond test of each die is equal to whatever maximum bandwidth is available from the tester to that particular die. That will depend on how many probe pads are available on the die. For post-bond test, the layers are tested in parallel under the constraint that the total amount of bandwidth for all layers is equal to the bandwidth between the tester and the pins of the bottom layer which is then distributed to the non-bottom layers via test elevator TSVs. The final post-bond test time is the largest test time among all layers in the design. The test data bandwidth that should be allocated to each layer to minimize the total test time needs to be selected.

The problem of determining the optimal post-bond bandwidth allocation to each layer can be formulated and solved with dynamic programming. The dynamic programming formulation can be written as follows:

$$\begin{cases} \mathrm{T}(D_{i\ldots k}, B) = \min\{\ \max[\ time_i(b_i), T(D_{i+1\ldots k}, B - b_i)\ ]\ \} \\ \qquad\qquad if\ i < k \\ \\ \mathrm{T}(D_{i\ldots k}, \mathrm{B}) = time_i(B) \\ \qquad\qquad if\ i = k \end{cases}$$

Where $D_{i\ldots k}$ represents set of dies from i through k. $T(D_{i\ldots k}, B)$ means the test time for dies from i through k with bandwidth B, b_i must be greater than 0 and represents the

bandwidth allocated to core *i*, and $time_i(B)$ represents the test time for *die i* with bandwidth *B*.

The dynamic programming formulation above suggests that the search for an optimal bandwidth allocation can be divided into recursive subproblems. In addition, instead of examining all possible combinations exhaustively, the algorithm efficiently stores the results of overlapping subproblems into a table to avoid repeating unnecessary computation.

Function $time_i(B)$ is a design-dependent function which queries what the test time for a layer would be using a given bandwidth *B*. To get this number, one of the many 2D algorithms for optimizing the test architecture for a given test data bandwidth can be used. For example, the TR-Architect algorithm [Goel 03] could be used as was done in the experiments reported in Section 4, but there are many others in the literature [Xu 05], [Larsson 05]. The proposed approach is applicable regardless of how the test architecture is optimized on each layer.

When determining $time_i(B)$, the existence of the bandwidth adapter needs to be taken into consideration. If *B* is smaller than the largest bandwidth used for die *i* in any phase of test, which will be referred to as $B_{i,max}$ then for bandwidth *B*, the bandwidth adapter will be used to convert bandwidth *B* into the larger bandwidth $B_{i,max}$. The test time in this case will be equal to $time_i(B_{\mathrm{i,max}})$ scaled for the delay through the bandwidth adapter:

$$time_i(B) = [time_i(B_{i,\max}) - P](\frac{B_{i,\max}}{B_i}) + P \quad when\ B < B_{i,\max}$$

where *P* is the number of test patterns applied by the bottleneck TAM. The capture cycles for these patterns should not be scaled, so they are subtracted out before multiplying by the scaling factor so that only the shift time is scaled, and then the capture cycles are added back in unscaled.

The following is an illustrative example of deriving the optimal bandwidth allocation for a post-bond test. There are four dies in this example, *D0, D1, D2* and *D3* to

be placed at layer 0, layer 1, layer 2, and layer 3. The post-bond bandwidth is 6 in this example. Table 1 shows the relationship between bandwidth and test length for each die, namely the value of $time_i(B)$. Note that each die may come from very different designs, so the relationship between test time and bandwidth may not be consistent.

Table 1. Example of $time_i(B)$

B_i	D0	D1	D2	D3
1	40	8	20	5
2	30	7	20	5
3	20	6	10	5
4	10	5	10	5

The algorithm begins with checking $T(D_{0...3}, 6)$, which creates subproblems $T(D_{1...3}, 5)$, $T(D_{1...3}, 4)$, $T(D_{1...3}, 3)$, $T(D_{1...3}, 2)$ and $T(D_{1...3}, 1)$. An example of an overlapping subproblem is that both $T(D_{1...3}, 3)$ and $T(D_{1...3}, 2)$ produce subproblem $T(D_{2...3}, 1)$. Only necessary computation will be performed and will only be performed once. In this example, the optimal allocation *is {b0, b1, b2, b3} = {3, 1, 1, 1}*. The post-bond test time with the derived bandwidth allocation is 20.

Now consider partial stack tests. The bandwidth allocation problem can be solved in the same way as it is for final stack except that there are fewer layers. The same dynamic programming approach can be used. The key is to order the processing from smallest stack to largest stack because the bandwidth allocation for smaller stacks will be larger than for larger stacks. In this manner, $B_{i,max}$ will be determined right away and then in all subsequent steps, the use of the bandwidth adapter will be known up front and can be factored in when determining $time_i(B)$.

So the overall procedure is to first determine the bandwidth for each die during pre-bond test. This is a given based on the number of probe pads available on the die. Next, if partial stack testing is to be done, then the bandwidth allocation for the smallest partial stack is computed first (using dynamic programming) since it will use the largest bandwidths, followed by the next smallest partial stack, and so forth until finally doing the full stack. The test architecture for each die is then designed and optimized based on the maximum bandwidth, $B_{i,max}$, that it receives in any phase of test. An input and output bandwidth adapter is then added to handle all other bandwidths that the die will see in any phase of test.

2.3. EXPERIMENTAL RESULTS

Experiments were performed on the ITC'02 benchmarks. [Marinissen 02] provides detailed descriptions of the ITC'02 benchmarks. Each 3D benchmark has four layers and each layer contains a design from the ITC'02 benchmark set. Table 2 lists the components of each 3D benchmark.

Table 2. 3D Benchmarks

Design	Layer 1	Layer 2	Layer 3	Layer 4
1	h953	d695	u226	d281
2	f2126	g1023	d695	u226
3	p93791	p34392	p22810	g1023
4	q12710	p34392	p22810	f2126
5	p93791	q12710	p34392	p22810
6	t512505	p93791	q12710	u226
7	a586710	p93791	q12710	p22810
8	a586710	t512505	p93791	p34392

To derive the test time for each design for different test data bandwidths, TR-Architect [Goel 03] was used in two scenarios. One is where the scan architecture is assumed fixed in the design as would be the case for hard cores (results shown in Table 3), and the second is where the scan architecture is assumed to be flexible as may be the case for soft cores (results shown in Table 4). The pre-bond bandwidth was assumed to be 8 in these experiments for the non-bottom layers, and the bottom layer was assumed to have a test data bandwidth of 32, 48, or 64. All possible partial stack tests are tested. The first partial stack test, which test layer 0 and layer 1 is referred to as intermediate test 1, and the second partial test is referred to as intermediate test 2. The final stack test involves testing all dies together.

In Table 3 and 4, results are shown for 3D designs 1 through 8 which are constructed with one ITC'02 benchmark circuit on each layer corresponding to what is shown in Table 2. The test time (in clock cycles) for each phase of test is shown comparing two cases. One is where a separate test architecture is designed for each different bandwidth (i.e., no bandwith adapter is used), and the other is where a single unified test architecture is designed using a bandwidth adapter.

As can be seen from the results in Tables 3 and 4, the overall test time of the proposed approach with a unified test architecture is very close to that of having separately optimized test architectures. In fact, it is even better in a few cases, but that is likely due to variations in the efficiency of different bandwidths for a design. The result is very good because it demonstrates that a unified test architecture which is simpler to design and requires less hardware overhead is able to do almost as well as if one custom designed a test architecture for each bandwidth that a layer sees.

Table 3. Experimental Result for Non-Flexible Designs

	Pre-bond Bandwidth	Post-Bond Bandwidth	**Pre-bond**		**Intermediate Test 1**		**Intermediate Test 2**		**Final Stack**		**Overall**		
			Separately Optimized	Unified w/Adapter	Separately Optimized	Unified w/Adapter	Separately Optimized	Unified w/Adapter	Separately Optimized	Unified w/Adapter	Separately Optimized	Unified w/Adapter	**Diff.**
1	8	32	267163	267163	132343	132343	132343	132343	132343	132343	664192	664192	100.00%
	8	48	267163	267163	132343	132343	132343	132343	132343	132343	664192	664192	100.00%
	8	64	267163	267163	132343	132343	132343	132343	132343	132343	664192	664192	100.00%
2	8	32	537887	537887	357757	357757	357757	357757	357757	357757	1611158	1611158	100.00%
	8	48	537887	537887	357757	357757	357757	357757	357757	357757	1611158	1611158	100.00%
	8	64	537887	537887	357757	357757	357757	357757	357757	357757	1611158	1611158	100.00%
3	8	32	3961050	3851545	1314769	1314769	1522806	1526525	1538204	1533345	8336829	8226184	98.67%
	8	48	3567417	3670068	921136	921136	1042640	1024827	1068036	1062649	6599229	6678680	101.20%
	8	64	3340565	3477831	695066	695066	812318	795918	833549	812143	5681498	5780958	101.75%
4	8	32	6763984	6763984	3466238	3466238	3466238	3466238	3466238	3466238	17162698	17162698	100.00%
	8	48	6763984	6763984	3466238	3466238	3466238	3466238	3466238	3466238	17162698	17162698	100.00%
	8	64	6763984	6763984	3466238	3466238	3466238	3466238	3466238	3466238	17162698	17162698	100.00%
5	8	32	9233954	9578467	3466238	3466238	3466238	3466238	3466238	3466238	19632668	19977181	101.75%
	8	48	9233954	9578467	3466238	3466238	3466238	3466238	3466238	3466238	19632668	19977181	101.75%
	8	64	9233954	9578467	3466238	3466238	3466238	3466238	3466238	3466238	19632668	19977181	101.75%
6	8	32	30388711	30388711	22973206	22973206	23162552	23699371	24703920	23699371	101228389	100760659	99.54%
	8	48	22627954	22627954	15212449	15212449	17126882	15808671	17229905	15883686	72197190	69532760	96.31%
	8	64	20351239	20351239	12935734	12935734	13024248	13486929	13024248	13486929	59335469	60260831	101.56%
Avg													100.95%

Table 4. Experimental Result for Flexible Designs

	Pre-bond Bandwidth	Post-Bond Bandwidth	**Pre-bond**		**Intermediate Test 1**		**Intermediate Test 2**		**Final Stack**		**Overall**		
			Separately Optimized	Unified w/Adapter	Separately Optimized	Unified w/Adapter	Separately Optimized	Unified w/Adapter	Separately Optimized	Unified w/Adapter	Separately Optimized	Unified w/Adapter	**Diff.**
1	8	32	185043	193729	58383	58383	66955	66656	68623	68512	379004	387280	102.18%
	8	48	165924	175251	39264	39264	44455	45225	47749	48264	297392	308004	103.57%
	8	64	155826	162168	29165	29165	33957	33794	35351	35554	254299	260681	102.51%
2	8	32	348445	348445	179417	179417	200231	197433	216881	200047	944974	925342	97.92%
	8	48	287954	287954	118926	118926	134129	134104	141317	137619	682326	678603	99.45%
	8	64	259052	259052	90024	90024	99066	99066	104303	102295	552445	550437	99.64%
3	8	32	3818236	3875858	1290998	1290998	1514161	1514161	1531433	1570649	8154828	8251666	101.19%
	8	48	3416782	3535277	885071	885071	1006965	1016655	1037496	1042895	6346314	6479898	102.10%
	8	64	3223401	3428306	691690	691690	777193	787404	787323	809828	5479607	5717228	104.34%
4	8	32	4348203	4416227	1227133	1227133	1443224	1443023	1633317	1578602	8651877	8664985	100.15%
	8	48	3940850	4129643	842227	842227	971816	982984	1068114	1082651	6823007	7037505	103.14%
	8	64	3753565	4028200	632495	632495	747585	746656	819780	831156	5953425	6238507	104.79%
5	8	32	7106302	7105983	1633317	1633317	2040990	2057327	2227321	2228377	13007930	13025004	100.13%
	8	48	6616992	6665707	1085050	1085050	1406248	1406248	1529900	1533397	10638190	10690402	100.49%
	8	64	6365179	6463831	833237	833237	1047365	1068134	1169749	1179100	9415530	9544302	101.37%
6	8	32	29922077	29922077	22973206	22973206	24263686	24437740	24333128	24477405	101492097	101810428	100.31%
	8	48	21973150	21973150	15024279	15024279	16249811	15620048	16320517	15938714	69567757	68556191	98.55%
	8	64	18695162	18695162	11746291	11746291	12532429	12138131	12553054	12138131	55526936	54717715	98.54%
Avg													101.13%

Note that even though the tester bandwidth is the same for all cases in pre-bond test, some cases reports different pre-bond test times. This arises due to the bandwidth adapter having to convert from a smaller tester bandwidth to a larger internal TAM bandwidth during pre-bond tests. It can be harder to efficiently schedule cores for larger TAM bandwidth. However, the test time in post-bond test is limited by the layer that takes the most time since post-bond test is performed in parallel. Therefore, the dynamic programming algorithm will try to use a larger TAM bandwidth for the bottleneck layer which may slightly increase pre-bond test time due to the inefficiency mentioned above. For non-bottleneck layers, the algorithm always try to use a bandwidth bigger than pre-bond bandwidth to avoid conversion at pre-bond. However, the net effect is improved overall test time because the reduction in post-bond test time is significantly greater than the increase in pre-bond test time.

2.4. CONCLUSIONS

The proposed approach for designing a unified test architecture across all test stages in testing 3D-ICs allows efficient hardware utilization without significant overhead in test time. It is simple to design and implement, and can be used together with any of the existing 2D test architecture optimization schemes.

3. Improving Test Compression with Multiple-Polynomial LFSRs

Increasing integration density coupled with the need for more types of tests to be applied to achieve quality requirements has resulted in rapidly increasing test data volume. Test data compression is widely used to compress the amount of data stored on the tester. This helps to reduce tester storage requirements and improve test time as less data has to be transferred over the limited test data bandwidth between the tester and chip-under-test [Touba 06].

One highly efficient approach for compressing test cubes (which are test vectors in which the inputs unassigned by ATPG are left as don't cares) is to use sequential linear decompressors to encode them. Data from the tester are injected into a linear feedback shift registers (LFSR) while it loads scan chains thereby "dynamically" reseeding it. The final value of each scan cell after decompressing a test cube can be written as a linear equation in terms of the bits coming from the tester which act as free variables that can be assigned any value. Test cubes can be encoded by solving a system of linear equations where each equation corresponds to a care bit in the test vector being encoded [Könemann 01], [Krishna 01], [Rajski 04].

The amount of test compression achieved can be increased by increasing the number of scan chains driven by the decompressor (i.e., increasing the "expansion ratio'). As the expansion ratio is increased, it becomes increasingly difficult to solve the system of linear equations because more care bits need to be encoded while the number of free-variables remains the same. At some point, the equations for two or more care bits will become linearly dependent and not be solvable. The bottleneck tends to be the scan cells loaded the earliest because they depend on fewer free-variables as the number of free-variables injected in the LFSR in the beginning is small. Consequently there is less encoding flexibility for solving the scan chains loaded the earliest and they will tend to limit the overall test compression that can be achieved.

One powerful way to increasing the encoding flexibility is to use a multiple-polynomial LFSR [Hellebrand 95] in which there are several choices for the characteristic polynomial of the LFSR. The polynomial used for encoding each test cube can be selected. So if the system of linear equations for one polynomial is linearly dependent and unsolvable, then a different polynomial can be selected to change the linear correlations and provide a different system of linear equations that may be solvable. If the multiple-polynomial LFSR can be configured with say 16 different feedback polynomials, then there are 16 different sets of linear equations which can be solved thereby providing a much higher probability of encoding a test cube. This makes a big difference particularly for the scan cells loaded the earliest which depend on fewer free variables. By adding additional flexibility in the selection of the polynomial, this weakness can be significantly relieved allowing higher expansion ratios to be used to achieve greater test compression.

The original idea of using a multiple-polynomial LFSR to help in encoding test cubes was originally proposed in [Hellebrand 95] in the context of static LFSR reseeding in a built-in self-test (BIST) environment. In static LFSR reseeding, only the initial seed of the LFSR contains free-variables. No additional free-variables are injected during decompression. Thorough analysis and results are presented in [Hellebrand 95] showing the effectiveness of multiple polynomials in reducing the number of bits that need to be stored on-chip compared with using a single polynomial LFSR for encoding test cubes with static reseeding in a BIST environment.

In this chapter, the use of multiple-polynomial LFSRs for dynamic reseeding in a test compression environment is investigated. The new contributions of this work include the following:

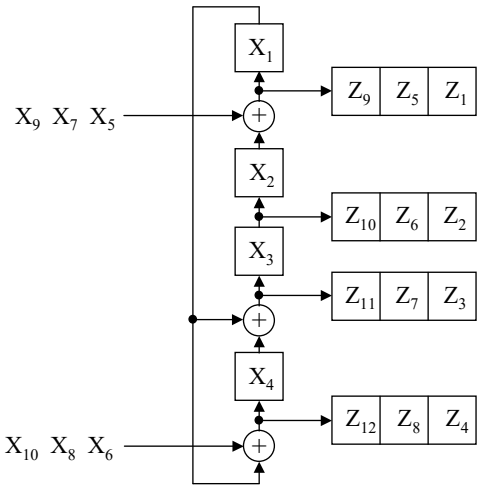
- A scheme for using multiple-polynomial LFSRs without requiring any control data by formulating it as a matching problem
- A way to implement a multiple-polynomial LFSRs with less overhead by exploiting properties of primitive polynomials

- A method to use multiple-polynomials to increase the effectiveness of retaining unused free-variables [Muthyala 12]
- Experimental results demonstrating the improvements that can be obtained by using multiple-polynomials in dynamic reseeding alone and with retained free-variables.

The chapter is organized as follows: Section 3.1 reviews the fundamentals of sequential linear decompression. Section 3.2 describes the hardware implementation of the multiple polynomial LFSR with rotating polynomials. Section 3.3 presents the algorithm to assign test cubes to polynomials. Section 3.4 shows the experimental results. Section 3.5 is a conclusion.

3.1. SEQUENTIAL LINEAR DECOMPRESSION

This section reviews the fundamental concepts behind sequential linear decomposition and techniques for reducing tester data through retaining free variables. In sequential linear decompression, variables are continuously injected into a LFSR [Mrugalski 04]. To determine the value of each injected variables, Linear equations can be obtained thorough symbolic simulation of the linear decompressor. This process is illustrated in figure Figure 6. At each cycle, new variables are injected into the decompressor. In Figure 6, these injected variables are represented by *X*. Also at each cycle, the content of the LFSR are being sent to the scan chains. These variables on the scan chains are represented by *Z*. At the bottom of Figure 6, a table is given for the linear equations of the variables after three cycles of symbolic simulations.



$Z_9 = X_1 \oplus X_4 \oplus X_9$	$Z_5 = X_3 \oplus X_7$	$Z_1 = X_2 \oplus X_5$
$Z_{10} = X_1 \oplus X_2 \oplus X_5 \oplus X_6$	$Z_6 = X_1 \oplus X_4$	$Z_2 = X_3$
$Z_{11} = X_2 \oplus X_3 \oplus X_5 \oplus X_7 \oplus X_8$	$Z_7 = X_1 \oplus X_2 \oplus X_5 \oplus X_6$	$Z_3 = X_1 \oplus X_4$
$Z_{12} = X_3 \oplus X_7 \oplus X_{10}$	$Z_8 = X_2 \oplus X_5 \oplus X_8$	$Z_4 = X_1 \oplus X_6$

Figure 6. Symbolic Simulation of Sequential Linear Decompression

Figure 7 is a Boolean matrix representation of the linear equations obtained from the LFSR in Figure 6. Solving these linear equations determines what value should be assigned to each variable. Based on the equations, some of the variables may need to be a certain value, whereas some variables may be freely assigned with any value.

$$\begin{pmatrix} 0&1&0&0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0 \\ 1&0&0&1&0&0&0&0&0&0 \\ 1&0&0&0&0&1&0&0&0&0 \\ 0&0&1&0&0&0&1&0&0&0 \\ 1&0&0&1&0&0&0&0&0&0 \\ 1&1&0&0&1&1&0&0&0&0 \\ 0&1&0&0&1&0&0&1&0&0 \\ 1&0&0&1&0&0&0&0&1&0 \\ 1&1&0&0&1&1&0&0&0&0 \\ 0&1&1&0&1&0&1&1&0&0 \\ 0&0&1&0&0&0&1&0&0&1 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \\ X_8 \\ X_9 \\ X_{10} \end{pmatrix} = \begin{pmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \\ Z_5 \\ Z_6 \\ Z_7 \\ Z_8 \\ Z_9 \\ Z_{10} \\ Z_{11} \\ Z_{12} \end{pmatrix}$$

Figure 7. Matrix Representation of Linear Equations

The process of solving the linear equations is illustrated in Figure 8. A submatrix which corresponds to the rows that specify Z as care bits are extracted. The submatrix is then solved with Gauss-Jordan elimination. After the Gauss-Jordan elimination, variables can be put into two different categories: pivot and non-pivot variables. The pivot variables represent the variables that need to be assigned to a particular value to encode the test cube whereas the non-pivot variables can be freely assigned with any value. Therefore, non-pivot variables are also called free variables. It is desirable to increase the number of free variables to reduce amount of data required to be stored on tester.

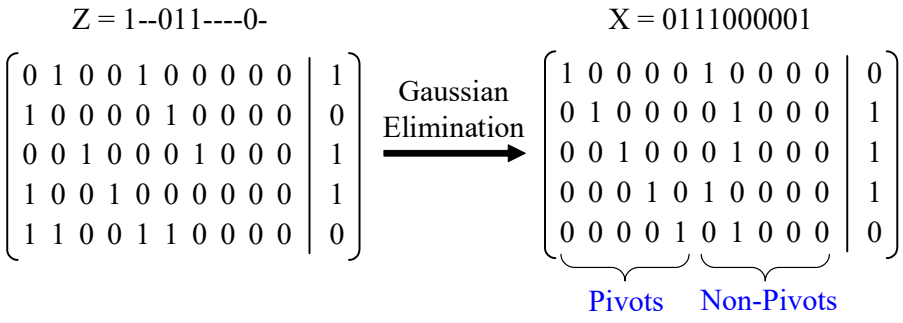


Figure 8. Solving Linear Equations

Approaches to increase the number of free variables have been proposed. In [Muthyala 12], a FIFO is added before the sequential decompressor to retain the free variables. The retained free variables can then be used to encode the next test cube. It has been shown that using a FIFO can help to reduce amount of tester data with minimal performance impact.

3.2. PROPOSED APPROACH

The concept of the proposed scheme is instead of having a fixed LFSR structure with one polynomial, the LFSR can be reconfigured to implement multiple different polynomials. If a test cube is not encodable by a particular polynomial, a different polynomial can be used instead.

A Galois LFSR is a type of LFSR where the last bit is sent back and XORed with shifted values at certain tap locations. The tap locations depend on the polynomial of the LFSR. A multiple polynomial LFSR is implemented with reconfigurable tap locations. Figure 9 shows a Galois multiple-polynomial LFSR for sequential linear decompression with k bits. The TAP_i signals in Figure 9 select which locations to feedback the last bit B_k from the LFSR to implement the desired polynomial.

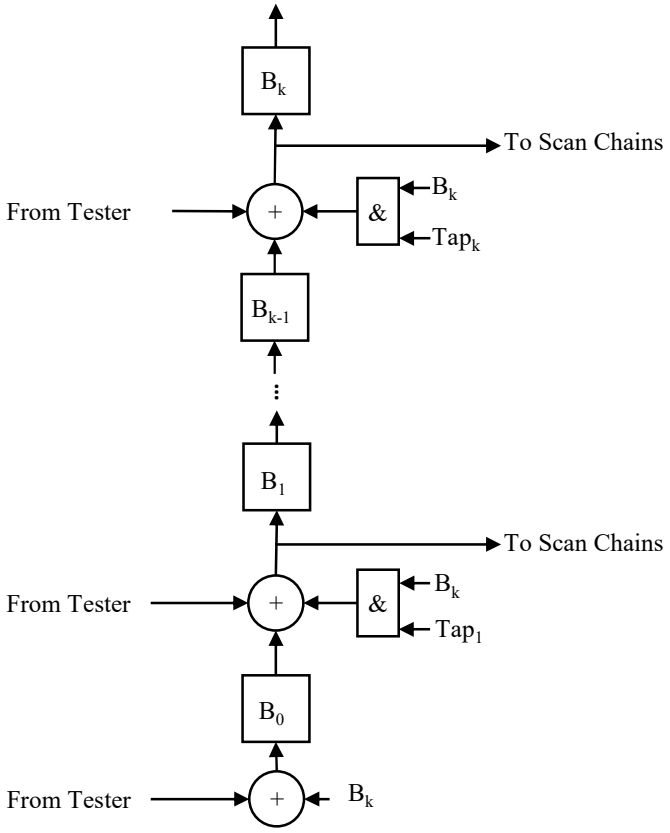


Figure 9. Multiple-polynomial LFSR

If the multiple polynomial LFSR is to be used with a set of p polynomials, then $\log_2\lceil p \rceil$ control bits are needed to select which polynomial to use. The p polynomials could either be stored in a ROM with p address locations each storing k-1 bits for a k-bit LFSR (it is k-1 because the first tap point is always used for any polynomial), or they could be implemented with combinational logic having $\log_2\lceil p \rceil$ inputs. The advantage of using a ROM implementation is that the set of polynomials can be chosen after the test cubes are generated.

In dynamic LFSR reseeding, where continuous flow decompression is used, it is undesirable to have to supply additional control bits to select the polynomial when

decompressing a test cube. The proposed approach solves this problem by exploiting the fact that test cubes can be reordered. The proposed approach uses a counter to select the polynomial each time a test cube is decompressed such that the polynomials are constantly rotating (i.e., a different polynomial is used in each subsequent test cube decompression until all *p* polynomials have been used at which point it cycles back to the first one). In this way, the polynomial that is used to encode a particular test cube can be selected by the way the test cubes are ordered. Many test cubes can be encoded by all the polynomials, so it doesn't matter which one they are matched with. For some test cubes, only some of the polynomials can be used to encode them, so they must be placed in spots where they will be decompressed with a polynomial that can encode them. The next section will formulate the problem of ordering the test cubes as a bipartite matching problem. The advantage of this approach is that no control bits are needed for selecting the polynomial, so there is no additional data that needs to be stored and transferred from the tester to use a multiple polynomial LFSR compared to what is used for a conventional single polynomial LFSR.

Figure 10 shows the proposed hardware implementation of decompression with a multiple polynomial LFSR with a rotating polynomial. In each new test cube decompression, the counter increments to switch to the next polynomial. The circuitry that generates the polynomial can be either a combinational decoder or a ROM. If it is desired to use the scheme described in [Muthyala 12] to retain the unused free variables for one test cube to help in encoding the next test cube, a FIFO can optionally be added. This helps to increase encoding efficiency. To maximize the number of free variables retained in the FIFO, the process that assigns the test cubes to polynomials can be modified as will be described in the next section.

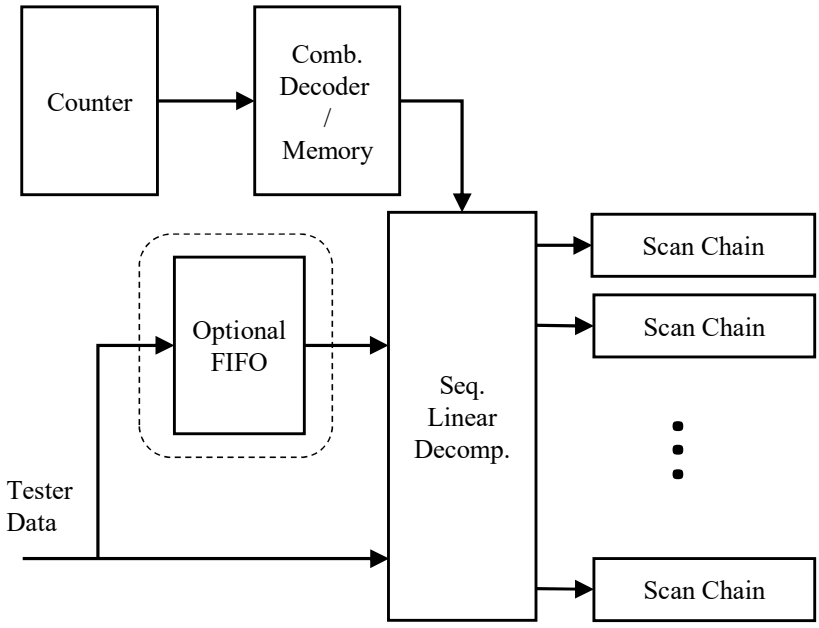


Figure 10. Proposed decompression scheme with multiple-polynomial LFSR

Hardware overhead for the multiple polynomial LFSR can be further reduced by exploiting a property of primitive polynomials which is that the reciprocal of a primitive polynomial is also primitive [Pless 11]. The reciprocal is formed by simply reversing the order of the coefficients in a polynomial. For example, for the primitive polynomial ($x^4 + x + 1$, i.e., coefficients 10011), the reciprocal polynomial is ($x^4 + x^3 + 1$, i.e, 11001) is also primitive. Our experiments showed that the set of test cubes that can be encoded by a primitive polynomial and its reciprocal polynomial is as diverse as the polynomial with an arbitrary different primitive polynomial. There is no noticeable degradation in using a reciprocal polynomial versus using any other arbitrary primitive polynomial. The idea proposed here is to use this property to reduce the hardware costs. If a multiple polynomial with p primitive polynomials is to be used, instead of storing all p polynomials in the ROM, only $p/2$ polynomials could be stored in the ROM, and then MUXes can be used to generate the reciprocal of these $p/2$ polynomials to generate another set of $p/2$ polynomials as shown in Figure 11. This approach still generates p total primitive polynomials, but it uses a ROM that is half as large. However, it also requires $k-1$ 2-to-1 MUXes to reverse the polynomials

for a *k*-bit LFSR. These reason why it is *k-1* instead of *k* is that there are *k*+1 coefficients for each polynomial, but the first and last coefficients are always 1, so they do not need to be reversed. Overall, the cost of the *k-1* MUXes is much less than *(p/2)(k-1)* ROM bits for sufficiently large *p,* and it does not scale as *p* is increased. For a combinational decoder implementation, using reciprocal polynomials also helps reduce the size of the combinational logic although the improvement is less than it is for a ROM implementation.

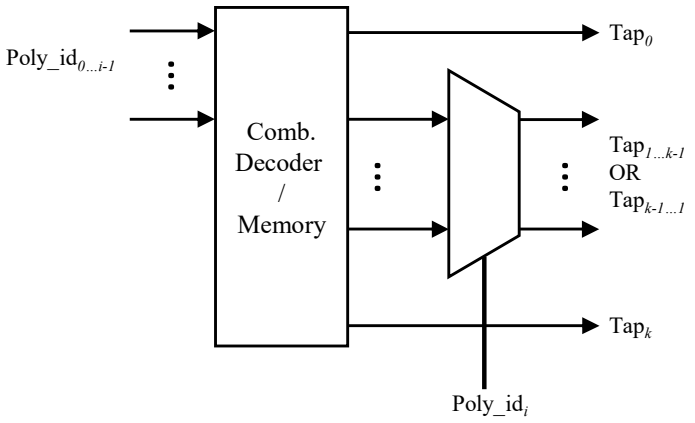


Figure 11. Reducing hardware overhead for primitive polynomial generation by using MUXes to select between a polynomial and its reciprocal

With the rotating polynomials, the test cube encoding process is the following. At design time, based on the length of the LFSR, *p/2* primitive polynomials of the correct length are arbitrarily selected (assuming the reciprocal polynomials will also be used). The decompressor is designed with a mod-*p* counter along with the selected polynomials without any assumption of a specific set of test cubes. Once the set of test cubes are generated after ATPG, the test cubes are ordered using the procedure described in the next section. Note that multiple test cubes can be decompressed with the same polynomial since the counter will cycle back to the same polynomial repeatedly during decompression.

3.3. ASSIGNING TEST CUBES TO POLYNOMIALS THROUGH BIPARTITE MATCHING

Section 3.2 described the decompressor hardware with continuously rotating polynomials. In this section, an algorithm is described for ordering the test cubes in a way that each test cube is decompressed using a polynomial that can encode it.

Given the set of polynomials that are rotating in the decompressor, along with the test cubes that need to be encoded, the problem can be formulated as a bipartite graph:

- For each polynomial, a vertex on the left side is created for each time the polynomial is repeated during decompression due to the counter cycling through it. This total number of vertices on the right hand size will be equal to the total number of test cubes to be decoded. The set of vertices that represent the polynomials is denoted as $\boldsymbol{V_L}$.
- For each test cube, a vertex is created on the right side. The set of vertices that represent the test cubes is denoted as $\boldsymbol{V_R}$.
- For each vertex in $\boldsymbol{V_L}$ and $\boldsymbol{V_R}$, create an edge if the corresponding polynomial in $\boldsymbol{V_L}$ can encode the corresponding test cube in $\boldsymbol{V_R}$.

After the above procedure, the problem of assigning test cubes to polynomials is represented with a bipartite graph. A *matching* is a subset of edges such that each node appears exactly one time at the edges in the matching. The problem of finding the polynomials to encode all the test cubes is equivalent to a bipartite matching problem with a constraint that every node must appear in exactly one match.

Figure 12 depicts an example of the bipartite matching problem. There are six test cubes, each can be encoded by a subset of the three given polynomials. To cover all test cubes, the polynomials are repeated twice. For example, vertex P_{0_0} represents the first appearance of polynomial 0, and P_{0_1} represents the second appearance of polynomial 0. Test cube 0, 1, and 2 are represented by vertices c_0, c_1 and c_2 respectively. For each polynomial, the encodable test cubes are listed at the bottom. An example of a matching is shown in bold edges in Figure 12. Note that there is more than one matching possible. As

long as every vertex is matched, it is considered a legal solution to encode all test cubes with the given polynomials.

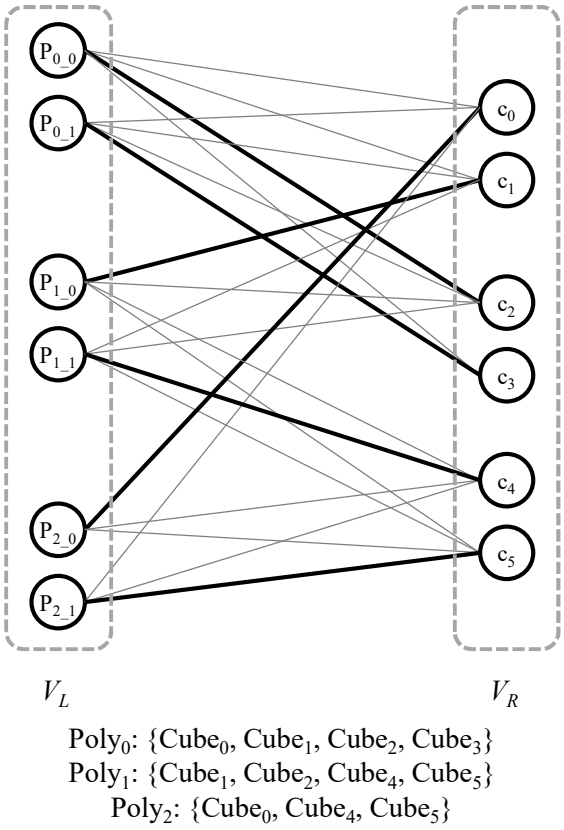


Figure 12. Example of assigning test cubes to polynomials thorough bipartite matching

Using the Ford-Fulkerson algorithm, the bipartite matching problem can be solved in *O(VE)*. In this case, the number of vertices is *O(T)* where *T* is the total number of test cubes, and the number of edges is *O(pT)* since each test cube can have up to *p* edges where *p* is the number of polynomials. Thus, the overall complexity will be $O(pT^2)$.

In Section 3, using a FIFO to implement the scheme in [Muthyala 12] was mentioned as an optional unit. The purpose of the FIFO is to improve encoding efficiency by retaining as many unused free variables as possible. Under this objective, in addition to matching test cubes to polynomials that can encode it, the number of free variables that can

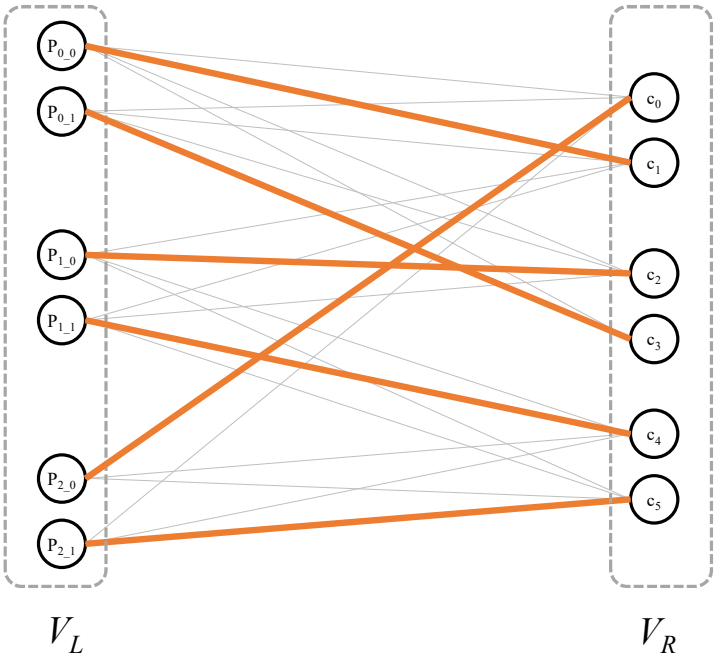
be retained should also be considered as this will vary depending on which polynomial is used to encode the test cube. This is because the FIFO only stores the latter free variables, and some polynomials will encode with more unused free variables towards the end than other polynomials will for a particular test cube. This can be considered by adding weights to the edges in bipartite graph corresponding to the number of free variables that get retained.

- For each edge between V_L and V_R, label the edge with weight equal to the number of free variable retained by encoding the test code represented by the vertex in V_R with polynomial represented by the vertex in V_L.

This revised scenario is equivalent to the maximum weighted bipartite matching problem. It has the same constraint that only one match is allowed for each vertex, but with additional objective that the optimal matching is the set of edges with the maximum sum of weights. Figure. 13 shows an example of building a maximum weighted bipartite matching problem and an optimal solution. The table on the left in Figure 13 describes the relationships between 6 test cubes and 3 polynomials. In the table, each entry shows how many unused free variables are retained when the polynomial is used to encode that test cube. If the entry is left blank, the polynomial cannot encode that test cube. On the right side of Figure 13, the edges between the vertices are created with the weight given in the table. For better readability, the weights on the edges are not shown. An optimal solution to the minimum-weighted matching problem is shown by bold edges in the graph.

	$Poly_0$	$Poly_1$	$Poly_2$
$Cube_0$	4	-	10
$Cube_1$	8	6	-
$Cube_2$	2	7	-
$Cube_3$	3	-	-
$Cube_4$	-	10	4
$Cube_5$	-	1	7

(a)



(b)

Figure 13. Assigning test cubes to polynomials considering number of free variables retained

The maximum weighted bipartite matching problem with non-negative edges can be solved in $O(V^2 \log V + VE)$. So in this case, the overall complexity will be $O(T^2 \log T)$ assuming the number of testcubes is much larger than the number of polynomials.

3.4. Experimental Results

A first set of experiments was performed using a 64-bit LFSR with 16 rotating primitive polynomials. The results are shown in Table 5. The first four columns show information about each circuit: number of test cubes, number of scan cells, and number of tester channels. Results are then shown for the conventional case where each test cube is encoded by the same polynomial. The number of scan chains was increased until it was no longer possible to encode all the test cubes with the given number of tester channels. The number of scan chains and the resulting amount of data that needs to be stored on the tester is shown for the conventional case.

Table 5. Results for Using Proposed Scheme to Encode Test Data

Circuit	Num. Vect.	Scan Cells	Tester Chans.	Conventional Single Poly. LFSR		Proposed						
						16 Poly. LFSR				16 Poly. LFSR + FIFO		
				Scan Chains	Tester Data	Scan Chains	Tester Data	Percent Reduction	Scan Chains	Tester Data	Percent Reduction	Overhead (Cell Area)
Ckt-A	266	3,828	5	27	190,190	35	151,620	25	37	139,225	37	35 FF
Ckt-B	540	5,020	8	88	272,160	108	224,640	21	118	207,360	31	46 FF
Ckt-C	490	6,370	7	54	404,187	71	312,214	29	73	298,872	35	44 FF
Ckt-D	592	7,417	10	95	485,440	110	402,560	21	130	361,120	34	47 FF
Ckt-E	711	8,742	9	81	697,491	111	537,516	30	112	506,331	38	32 FF
Ckt-F	615	12,225	6	60	771,210	75	619,920	24	78	597,708	29	31 FF

In the next section of Table 5, results were generated using the proposed multiple polynomial LFSR. The results under the major heading 16 poly. LFSR show the results without the optional FIFO. The number of scan chains was increased until it was no longer possible to encode all the test cubes with the given number of tester channels. As the number of scan chains goes up, the scan length goes down. Consequently, less data is shifted in from the tester to the decompressor, so the amount of data stored on the tester is reduced (i.e., the amount of test compression increases). The percentage reduction in test data is computed as:

$$\frac{(Original\ Test\ Data) - (New Test\ Data)}{(Original Test\ Data)}$$

The results show a 21-30% improvement in compression over conventional single polynomial LFSR when a rotating polynomial is used. This is because having multiple polynomials to choose from increases the chance of encoding a test cube, so more aggressive expansion to more scan chains is possible while still being able to solve the linear equations. That last columns in the table show the results for using an LFSR with 16 polynomials and including a FIFO to retain unused free variables using the method

described in [Muthyala 12]. As can be seen, this boosts the improvement in compression up to 29-39%. The overhead for the FIFO is shown in the last column in terms of the number of flip-flops used.

For the next set of experiments, the relationship between the number of polynomials used and the reduction in tester data was explored. The results are shown in Figure 14. Each of the circuits are shown on the *x*-axis with a bar for 1, 4, 8, and 16 polynomials. The amount of tester data used is shown on the *y*-axis. From the results, it can be seen that the more polynomials that are used, the more efficient the compression is. In some cases, the amount of compression stayed the same when the number of polynomials were increased (e.g., the results between 4 and 8 polynomials in Ckt-A), but then improved as the number of polynomials was increased further (i.e., the results between 8 and 16).

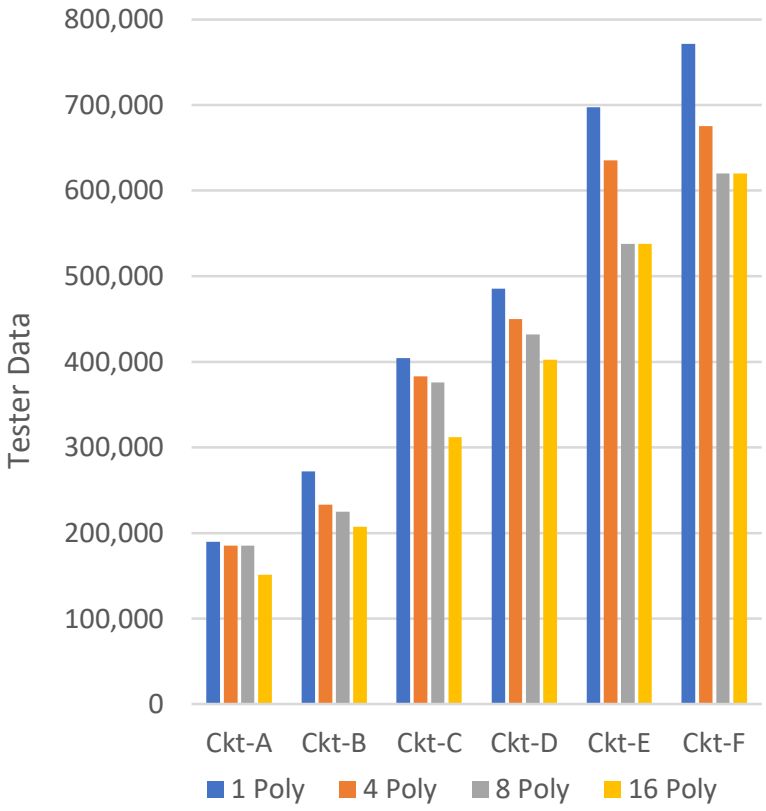


Figure 14. Number of Polynomials versus Test Data

3.5. CONCLUSIONS

Multiple polynomial LFSRs can significantly improve test data compression for sequential linear decompressors. This chapter proposed an efficient implementation of a multiple polynomial LFSR with rotating polynomials which exploits the degree of freedom in ordering test cubes to avoid the need for control bits. Using multiple polynomial LFSRs is a low cost way to boost test compression without creating any additional complexity for continuous-flow decompression. The only cost is the additional area overhead for the multiple polynomial LFSR versus single polynomials LFSR, and running the bipartite matching algorithm for ordering the test cubes which runs in polynomial time.

4. Improving Logic Obfuscation via Logic Cone Analysis[2]

Logic obfuscation has been proposed as a means to hide the functionality of a design to protect it from reverse engineering, hardware Trojan, and IP piracy [Roy 08, 10], [Chakraborty 09a], [Baumgarten 10], [Rajendran 12, 14]. One approach is to use combinational obfuscation where "key gates" are inserted in a design which have primary inputs that are referred to as "key inputs". The design will only function correctly if the correct key values are used. This approach was proposed in [Roy 08, 10]. The IP vendor activates the obfuscated design by storing the correct key values in a tamper-evident memory or using a physically unclonable function (PUF) [Suh 07] to generate them in a way that an attacker cannot access the key values. Without the key values, the design is unusable even if the netlist is obtained by either stealing the design or reverse engineering it.

Logic obfuscation with XOR and MUXes is investigated in [Rajendran 14]. In order to evaluate the effectiveness of the key bits, hamming distance (HD) was used to evaluate the result after application of a wrong key. 50% of HD is desired to ensure the malicious users will not be able to obtain correct output easily without the correct key.

In addition to raising the effectiveness of key insertion, [Rajendran 12] also pointed out the importance of protecting the key bits against “attacks”. If the attacker has the netlist and purchases a functional IC in the open market, then the attacker can try to determine the correct key input values to unlock the design. This is done by simulating input patterns on the netlist and comparing them with the correct output values obtained by running the same input patterns on the functional IC. It is shown in [Rajendran 12] that if an input pattern can be found which sensitizes a key input to a primary output without any interference from other key inputs, then the input pattern will propagate the correct key value to the primary output when running the pattern on the functional IC. The attacker can use ATPG

[2] The work in this chapter is published in [Lee 15]: Y.-W. Lee and N.A. Touba, "Improving Logic Obfuscation via Logic Cone Analysis," *Proceedings of IEEE Latin-American Test Symposium*, 2015.Yu-Wei Lee is the author and Nur. A. Touba is the supervisor.

to try to find an input pattern that sensitizes a key input to a primary output treating all other key inputs as X's. Each time such a pattern is found, the attacker runs it on the functional IC and looks at the output to resolve the key value. Whenever a key value is resolved, it is no longer an X when sensitizing the remaining key inputs thereby making the problem iteratively easier. After as many keys are resolved as possible using this approach, the remaining keys can be determined through brute force. The brute force procedure is to try each possible combination of values for the keys until the correct one is found. Checking if the candidate combination of key values is correct is done by simulating a set of random patterns on the netlist using the candidate key values and then comparing that with the output response obtained by simulating the same patterns on the functional IC to see if they match.

In [Rajendran 12], a heuristic procedure is proposed for inserting key gates so as to increase the amount of interference between the key gates and reduce the chance of finding patterns that sensitize a key input to an output without interference from other key inputs. The goal is to maximize the number of key values for which brute force must be employed thereby increasing the difficulty for the attacker. The heuristic procedure is based on constructing a graph in which each node is a key, and weighted edges are placed between the nodes to indicate the amount of interference that is created. Each key gate is iteratively inserted in the design so as to maximize the resulting sum of the weights on all edges in the graph at each step.

In this chapter, a new form of attack is considered where the attacker can reduce the complexity of brute-force attacks through analysis of logic cones. The ideas proposed here build on the concepts and overall framework introduced in [Rajendran 12]. An attack strategy is proposed in which the attacker considers the design one logic cone at a time. It is shown that in many cases, the number of brute force key combinations that need to be tried by the attacker can be significantly reduced. A technique for improving the strength of the logic obfuscation with respect to the considered attack strategy is proposed. It is

based on inserting key gates based of MUXes to increase the size and overlap of logic cones. The work in this chapter is published in [Lee 15].

MUXes were used in [Charkraborty 09b] to introduce additional inputs in order to protect hardware at system level. In the proposed idea, MUXes are used to avoid the situation where the attacker can take advantage of less secure logic cones to easily resolve key values and thereby iteratively attack the overall logic obfuscation. The MUX key gate insertion is done in a non-deterministic manner so that even if an attacker knows the key insertion algorithm, the attacker cannot exploit this knowledge to decipher the logic obfuscation.

4.1. BRUTE FORCE ATTACK STRATEGY BASED ON LOGIC CONES

The attack strategy proposed here can be applied after the key sensitization techniques proposed in [Rajendran 12] are applied to resolve as many keys as it can. For the remaining keys, the idea here is that brute force can be applied first to the logic cone containing the fewest number of key inputs, i.e., the least secure logic cone. All combinations of key inputs can be tried to identify for which combinations of values the simulated netlist output for that logic cone matches the functional IC output. In many cases, there will be only one combination that matches in which case all the key values for that cone are resolved. In some cases, there may be more than one combination of key values that matches, but it will likely be very few combinations which greatly reduces the search space for subsequent keys. Brute force is then applied for the next logic cone that has the fewest remaining unresolved key inputs. The process iterates through all the logic cones from easiest to hardest in terms of number of remaining unresolved key inputs in each cone. For each cone, additional keys are resolved or the number of possible solutions is greatly reduced which helps to simply brute force for the next logic cone.

This process is illustrated in Figure 15. The circuit has 6 key inputs and if brute force is used on the circuit as a whole, 26 different key combinations would have to be tried in the worst-case. However, output O_1 only depends on two key inputs (K_1 and K_2).

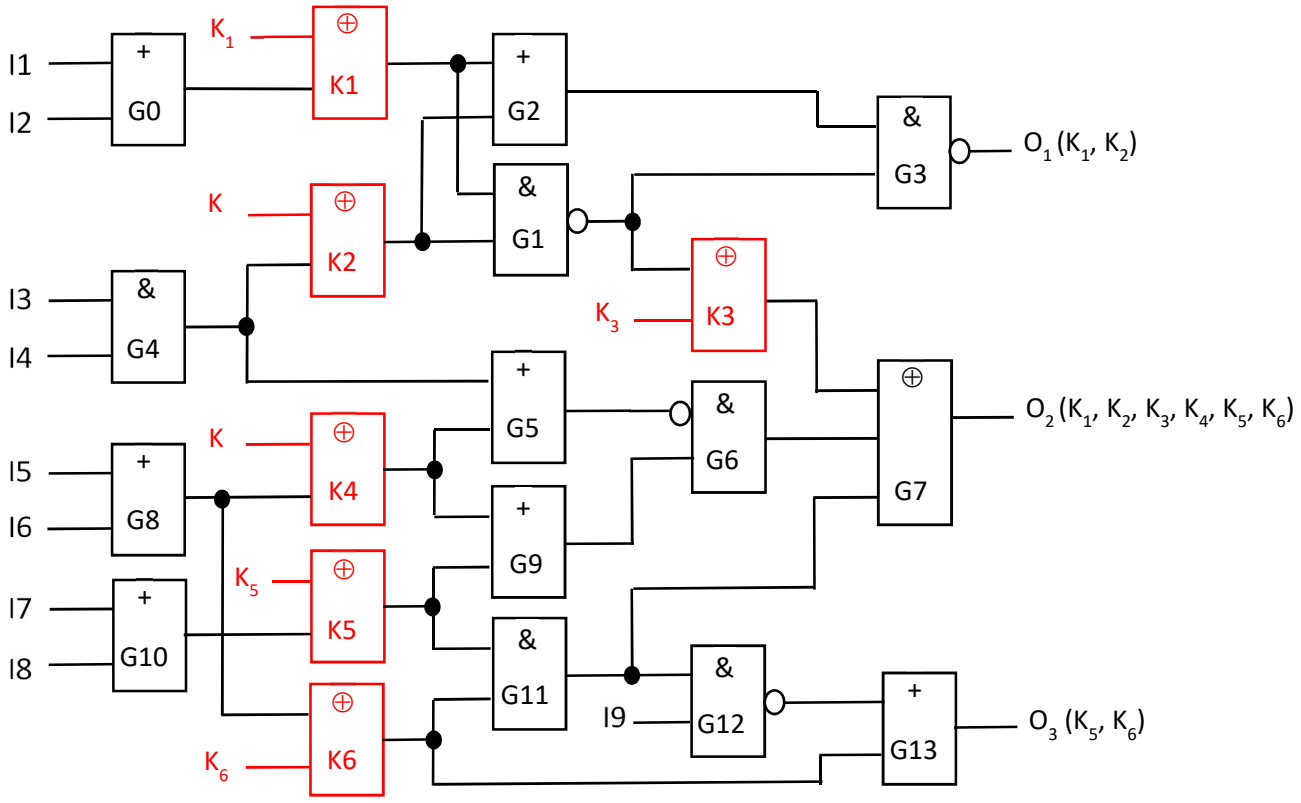


Figure 15. Example of Circuit with 6 Key Gates Inserted.

So brute force could be applied on that first requiring only 22 different key combinations in the worst-case to resolve K_1 and K_2. Next, output O_3 only depends on two key inputs (K_5 and K_6). Once those are resolved trying 24 combinations worst-case, then there are only two keys remaining (K_3 and K_4) for which brute force needs to be performed for O_2. So rather than 26 operations, only 24+24+24 operations would be required in the worst-case. While the difference in the number of operations is not large in this small example, it quickly scales up exponentially as the number of keys are increased.

4.2. Effective Key Size for Cone-Based Attack

As explained in detail in [Rajendran 12], there are a number of ways that the effects of a key can be "muted". For example, for some input patterns the output of a key gate can

be logically masked thereby reducing the number of keys that the circuit output depends on which helps to reduce the difficulty for the attacker. Different relationships between key gates such as dominating and convergent are studied in [Rajendran 12], and a procedure for constructing an interference graph is described. From the interference graph, the *effective key size*, which is the number of keys have to be solved through brute-force, can be computed. The number of brute force attempts required to decipher the keys is exponential in the effective key size. In this chapter, the procedure described in [Rajendran 12] for computing the effective key size from the defender's perspective (which is a conservative measure) is used as the basic metric for measuring security. The procedure for building the interference graph and computing this value is beyond the scope of this chapter, however, a detailed explanation of it can be found in [Rajendran 12].

The new aspect of this chapter is that the effective key size is computed w.r.t. each logic cone. Here we compute an effective key size against a cone-based attack as follows. It is assumed that the logic cone with the smallest effective key size is attacked first and its keys are resolved and removed from consideration when computing the effective key size for the remaining logic cones. From the remaining logic cones, it is assumed that the one with the smallest effective key size is attacked next and its keys are resolved. This process is then repeated iteratively until all keys are resolved. The logic cone with the largest effective key size when attacked defines the overall effective key size against a cone-based attack.

Once the attacker considered the logic cones, the computational complexity become much lower since the complexity is defined by the overall effective key size. Figure 16 presents an analysis of effective key size for a cone-based attack on four ITC99 benchmarks. The X-axis in Figure 16 shows the iterations of performing a cone-based attack. Squared markers on Y-axis show overall effective key size, and rounded markers show number of effective keys that is actually solved at each iteration. Initially, the overall effective key size is very high when all keys remain unresolved. With cone analysis, attackers find and solve a small cone first. The keys were removed subsequently and some

bigger cones may now contain fewer keys. This process was iteratively repeated until all the keys are resolved. As shown in Figure 16, during the entire attack process, the number of effective keys required to be solved remains much lower than the overall effective key size. It can be seen that the complexity is greatly reduced because having a cone that is easy to attack will also make larger cones become unsafe.

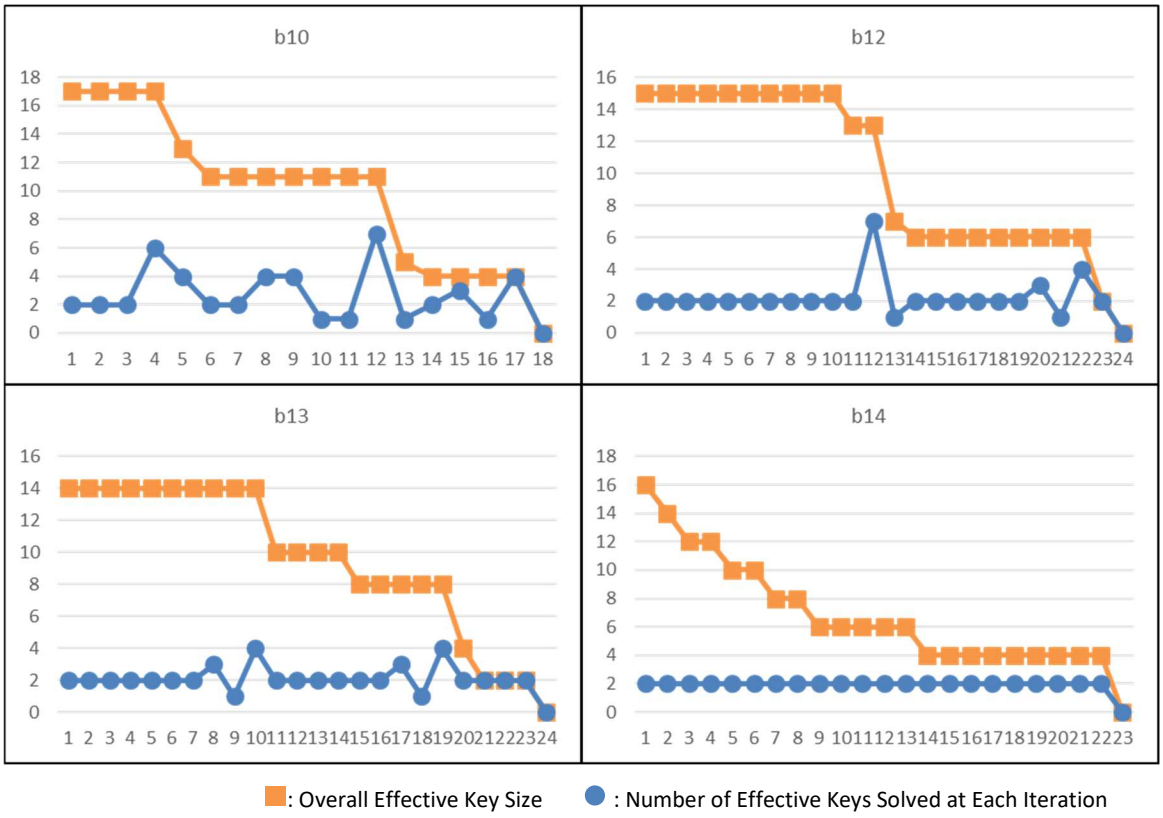


Figure 16. Analysis of Effective Key Size for Cone Based Attack

4.3. INSERTION OF KEY GATES TO COUNTER ATTACK

In order to counter the logic cone based attack strategy and ensure sufficient difficulty for the attacker to obtain the key values, it is necessary to consider the structure

of the logic cones in the circuit when inserting key gates. One way to improve the logic obfuscation would be to insert more key gates in a way that the number of key gates in each logic cone is sufficiently large. The number of logic cones is equal to the number of outputs which potentially could be quite high for large designs. Thus, it may require inserting a lot of key gates to get sufficiently high security for each and every logic cone.

The proposed approach involves inserting key gates based on MUXes. In this case, a MUX is inserted on some line in the circuit and the select line for the MUX is controlled by a key input. One data input for the MUX comes from the original line in the circuit, while the other data input comes from another line in the circuit which can be picked arbitrarily provided it does not create a feedback loop. The advantage of using MUX key gates is that they can increase the logic cone size and create more overlap between logic cones. This tends to increase the number of keys present in each logic cone thereby increasing the difficulty for an attacker using a cone-based attack. In fact, as will be seen in the experimental results, a relatively small number of MUX key gates can create a high degree of overlap among the output cones greatly reducing the effectiveness of a cone-based attack.

Figure 17 presents an example where two additional MUX key gates, K_7 and K_8, are inserted in the example from Figure 6. Both K_7 and K_8 expand the logic cones for O_1 and O_3 so that they include more keys. As a result, the overall effective key size is increased.

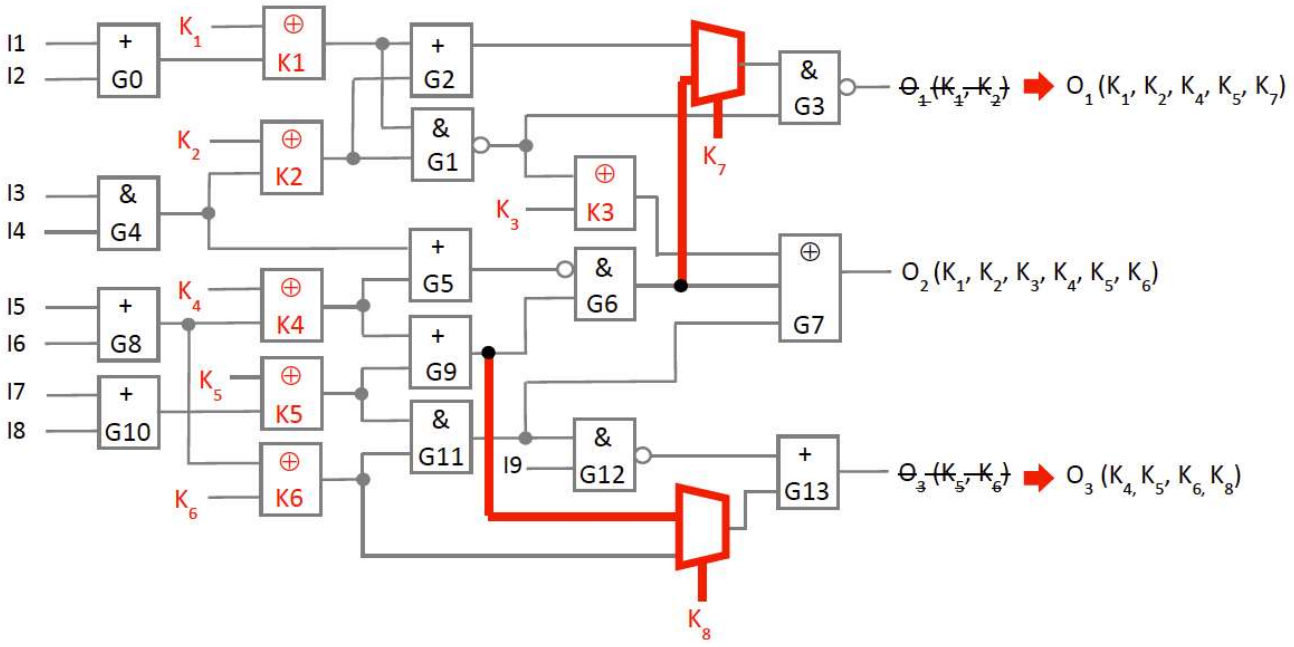


Figure 17. Example of Circuit with 2 MUX Key Gates Inserted

The effectiveness of each MUX key gate could be optimized by using an algorithm to carefully select the location of each MUX and the secondary data input for each MUX to maximize its impact. However, if the attacker knows the deterministic algorithm that is used for this, then the attacker may be able to exploit this information to decipher which input of the MUX is the real one and which is the artificial one. To avoid this, some randomness needs to be introduced.

Figure 18 presents pseudo code for selecting locations for MUX insertion. From the discussions in the previous sections, putting a MUX where the design is the most vulnerable as a key gate should help to reinforce the security. At the same time, the algorithm has to have some degree of randomness to prevent reverse-engineering of the algorithm. With these two goals in mind, the following is the proposed heuristic. First, identify the logic cone with the smallest effective key size in the design. Randomly choose

a node in that cone in which to insert the MUX key gate. Finally, connect the MUX so that normal functional signal flow goes through one data input and the other data input (i.e., side input) can be connected to an arbitrarily picked node that does not create feedback. Different strategies for picking the side input were investigated and will be discussed shortly.

```
Input: Design D
Output: Design D with one additional mux inserted

Function mux_insertion (design D)
   For all logic cone in design
      Find logic cone K with fewest effective keys
   Generate a random location L within cone K
   Create a new MUX gate at L
   Connect original gate L' with L
   Connect L with side_input( D, L )
End of mux_insertion
```

Figure 18. Pseudo Code for MUX Insertion

Experiments were performed to compare MUX key gate insertion with the proposed heuristic versus just inserting it randomly in the ITC99 benchmark circuits [Corno 00] in which 50, 80, and 100 keys were inserted. The results are shown in Table 6. The first group of results is the effective key size when the insertion was purely random. The second group of results is the effective key size when insertion location was randomly picked from the smallest logic cone in the design. Results are shown for the case where a mix of 50% MUX key gates and 50% XOR key gates are used, and for the case where all key gates are MUX

key gates. The side inputs was randomly selected regardless of the way the location was chosen for insertion. Results show that picking the insertion location randomly from the smallest logic cone greatly improves the effective key size over random selection. The results can be further improved if the side inputs to MUXes is carefully selected. The heuristic is discussed next.

Table 6. Comparison of Different MUX Input Selection Strategies in Effective Key Size

Circuit Name	Keys Insert	Random		Cone Size Based		Key Size Based	
		50% MUX	100% MUX	50% MUX	100% MUX	50% MUX	100% MUX
b07	50	10.4	16.1	16.3	24.5	11.4	14.6
	80	21.9	29.5	31.7	42.2	19.8	30
	100	24.2	39.6	37.6	50.5	26.2	47.8
b10	50	8.3	13.8	11.5	13.8	11.1	13
	80	15.7	27.2	38.2	43.5	18.2	25.8
	100	19.2	34.8	44.2	58.6	30.5	58.4
b11	50	7.2	9.6	11.1	18.5	9.7	18.9
	80	11.9	18.7	27.8	38.2	25.3	36.4
	100	15.2	25.2	42.1	53.2	41.2	49.9
b12	50	3.9	3.8	4.7	7	4.8	5.5
	80	5.3	7.6	9.9	12.9	8.7	13.1
	100	6.4	8.1	13.8	20.4	11.4	19.4
b13	50	5.5	6.4	9.9	11.3	8.7	10.5
	80	7.7	13.3	17.2	18.8	14.3	22
	100	10.5	17.1	23.4	32.5	21.1	28.9

Figure 19 presents pseudo code for picking side inputs for the MUXes inserted. After the location for inserting MUXes is determined, the algorithm randomly selects three candidate locations from all nodes circuit-wide from which to connect the secondary data input for the MUX (which would not create feedback). Finally, it uses a heuristic to select

one of the three candidates. The heuristic is implemented by the score function in Figure 19. The following discusses the two proposed heuristics.

```
Input: Design D, insertion location L
Output: Side input I for location L

Function side_input (design D, location L)
   While list C contains less than three locations
      Generate random location R within design D
      If (D,L) creates combinational Loop
         Discard R
      Else Add R to list C
   For each location Ci
      Calculate score (Ci)
   Return Ci with greatest score
End of side_input
```

Figure 19. Pseudo Code For Selecting Side Inputs

Generally speaking, it may be beneficial to select a side input coming from a large logic cone to effectively expand the input dependence of the logic downstream from where the MUX key gate is inserted. On the other hand, since the objective is to increase the effective keys, another strategy would be to pick a side input that itself depends on a large number of effective keys. Both of these heuristics were investigated. The following experiment implements and compares the two proposed heuristics. Similar to Table 6, experiments were performed for each of these heuristics on ITC99 circuits in which 50, 80,

and 100 keys were inserted. The results are shown in Table 7. The first set of results is for the case where pure random selection of the secondary data input of the MUX key gates is used (provided it doesn't create a feedback loop). The second set of results are for choosing the candidate with the largest number of gates in its cone. The last set of results are for choosing the candidate with the largest number of key gates in its logic cone. Results are shown for the case where a mix of 50% MUX key gates and 50% XOR key gates are used, and for the case where all key gates are MUX key gates. The results show that using the cone size based or key size based strategy is a significant improvement over random. In most cases, selecting the candidate that has the largest number of gates in its cone of logic proved to be more effective overall. This may be explained by the fact that because the number of keys is continually increasing as the key gate insertion process proceeds, the heuristic based on the number of keys has only partial information available at the time when it is used, and hence may not be as effective.

Table 7. Comparison of Different MUX Input Selection Strategies in Terms of Effective Key Size

Case	Keys	Random		Cone Size Based		Key Size Based	
		50% MUX	100% MUX	50% MUX	100% MUX	50% MUX	100% MUX
b07	50	10.4	16.1	16.3	24.5	11.4	14.6
	80	21.9	29.5	31.7	42.2	19.8	30
	100	24.2	39.6	37.6	50.5	26.2	47.8
b10	50	8.3	13.8	11.5	13.8	11.1	13
	80	15.7	27.2	38.2	43.5	18.2	25.8
	100	19.2	34.8	44.2	58.6	30.5	58.4
b11	50	7.2	9.6	11.1	18.5	9.7	18.9
	80	11.9	18.7	27.8	38.2	25.3	36.4
	100	15.2	25.2	42.1	53.2	41.2	49.9
b12	50	3.9	3.8	4.7	7	4.8	5.5
	80	5.3	7.6	9.9	12.9	8.7	13.1
	100	6.4	8.1	13.8	20.4	11.4	19.4
b13	50	5.5	6.4	9.9	11.3	8.7	10.5
	80	7.7	13.3	17.2	18.8	14.3	22
	100	10.5	17.1	23.4	32.5	21.1	28.9

Note that the use of MUX key gates has a little more routing complexity than XOR key gates because it is a three input gate versus a two input gate, so it may be advantageous to use a mix of XOR and MUX key gates. This is explored in the experiments presented in the next section.

4.4. EXPERIMENTAL RESULTS

Experiments were performed on the ITC99 benchmark circuits [Corno 00]. In Table 8, results are shown for inserting 50, 80, and 100 keys in each benchmark circuit. The first set of results are for computing effective key size across the whole circuit without

considering a cone-based attack. Area overhead was estimated with the 45nm FreePDK library [NCSU 11].

Table 8. Comparison of Effective Key Size

Name	Keys Inserted	Whole Circuit			Cone-Based						Area Overhead		
		Random Insertion			Random Insertion			Heuristic Insertion					
		0% MUX	50% MUX	100% MUX	0% MUX	50% MUX	100% MUX	0% MUX	50% MUX	100% MUX	0% MUX	50% MUX	100% MUX
b07	50	23.2	26.7	28.3	5.7	10.4	16.1	6	34.5	45.7	20.1%	20.9%	19.0%
	80	33.7	40.5	43.9	7.6	21.9	29.5	6.9	56.1	71.1	29.3%	28.7%	27.1%
	100	37	48.6	53	7.6	24.2	39.6	8.2	70.9	88.4	34.6%	33.8%	29.9%
b10	50	15.2	22.7	28.3	6.8	8.3	13.8	7	30.3	36.9	34.2%	33.2%	31.6%
	80	21.2	34.8	46.3	9.5	15.7	27.2	10.5	50.5	62.4	44.6%	43.8%	42.5%
	100	25	43.2	52.8	11.6	19.2	34.8	10.8	58.4	80.5	50.8%	50.7%	49.8%
b11	50	19.4	22.7	25.5	6.5	7.2	9.6	6.4	32.9	45.7	10.5%	12.2%	13.6%
	80	28.6	36.8	39.7	8.3	11.9	18.7	8.7	52.3	67.5	15.6%	17.6%	19.6%
	100	34.5	45.7	49.8	10.6	15.2	25.2	10.4	70.4	85.2	19.4%	21.4%	22.2%
b12	50	9.4	12.7	12	3.5	3.9	3.8	3.8	7.4	30.7	9.8%	9.6%	9.0%
	80	14.9	17.3	24.1	4.7	5.3	7.6	5	29.5	59.8	15.0%	14.7%	13.9%
	100	17	22.6	26.7	4.5	6.4	8.1	5.3	37.5	80	18.2%	17.3%	16.4%
b13	50	8.6	13.9	16.5	4.1	5.5	6.4	4.3	13.5	28.8	28.2%	27.5%	23.7%
	80	12	21.3	29.9	5.1	7.7	13.3	4.6	29	55	37.6%	37.6%	35.0%
	100	12.2	27.5	34.8	5.4	10.5	17.1	5.4	39.1	75	42.9%	42.8%	40.8%
b14	50	16.7	17.9	18.5	7.3	7.4	8.1	6.8	26.3	45.6	1.8%	1.8%	1.7%
	80	25.5	26.9	27.8	10.9	11.3	11.6	11	52.6	74.8	3.0%	2.8%	2.7%
	100	30.3	31.8	32.9	12.1	12.8	13.7	13.8	67.8	93.3	3.8%	3.5%	3.3%
b15	50	19.9	19.9	21.2	4.3	6	6.3	7.1	16.8	28.3	1.2%	1.1%	1.0%
	80	32.4	33.3	33.9	5.3	8.4	12.1	7.7	52.1	74	1.9%	1.8%	1.7%
	100	41.2	43.5	43.8	6.1	14	17.7	7.7	62	90.2	2.5%	2.2%	2.1%

Results are shown where different percentage mixes of keys based on XOR gates and keys based on MUXes are randomly inserted in the design (100% XOR, 50% XOR and 50% MUX, and 100% XOR). Because the results vary depending on where they key gates are inserted, the experiment was performed many times and the average effective key size across all experiments are reported in the table. In the second set of results, random insertion of key gates is used again, but the effective key size is computed for a cone-based attack (as described in Section 4.3).

As can be seen, the effective key size is much lower when the attacker employs the logic cone based attack described in this chapter. It is also clear to see that using key gates based on MUXes helps considerably to improve the effective key size as it causes the logic cones to become highly overlapped. The last set of results is for using the proposed heuristic non-deterministic procedure for inserting key gates. As can be seen, this approach does not help when only inserting XOR key gates, but if MUX key gates are inserted, then it is very effective in increasing the effective key size in comparison to random insertion. This is because it guides the insertion procedure to more efficiently overlap the logic cones to counter a cone-based attack.

Figure 20 shows the number effective keys when 100 key gates are inserted for circuit *b13* under different ratios of MUX and XOR key gates. The X-axis shows the percentage of MUXes inserted. The Y-axis shows the number of effective keys can be extracted after insertion. It is clear that the number of effective keys monotonically increased as the ratio of MUXes increased.

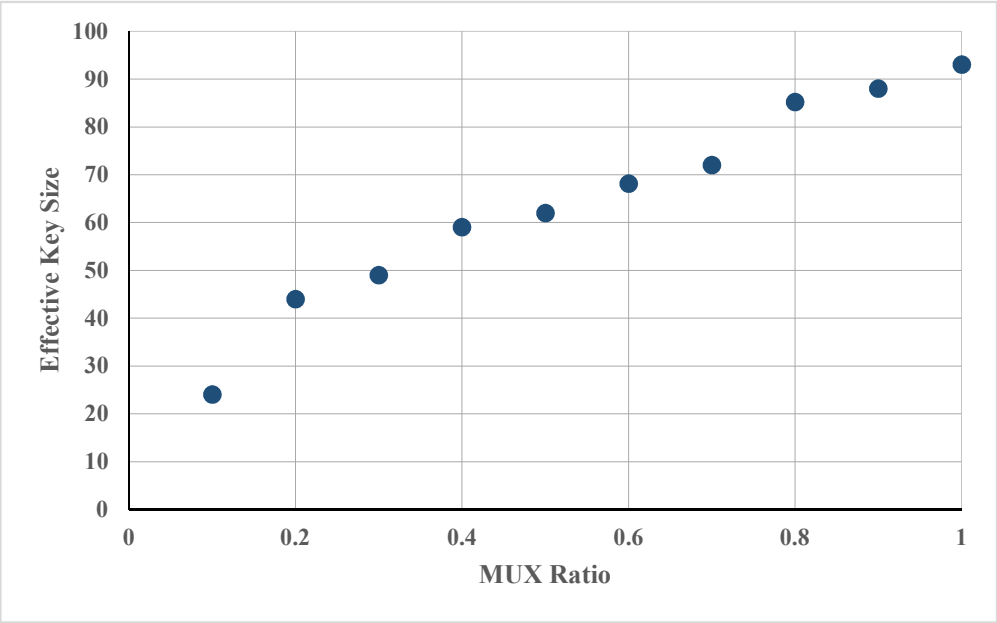


Figure 20. Effective Key Size for Cone-Based Attack for Insertion of 100 Keys in Circuit *b13*

Figure 21 shows a graph measure how many total gates need to be inserted to achieve an effective key size of 50 for a cone-based attack for circuit b11 for different ratios of MUX key gates and XOR key gates. The x-axis shows the percentage of MUX key gates that are inserted (with the rest being XOR key gates), and the y-axis shows the total number of key gates that need to be inserted to reach an effective key size of 50. As can be seen, as the percentage of MUX key gates is increased, fewer total key gates need to be inserted to achieve the same effective key size. However, the marginal improvement tapers off at around 40% of MUX gates. Since there is less overhead for inserting XOR key gates because they are 2-input gates versus MUX key gates that are 3-input gates, it may be cost effective to only insert 40% MUX key gates and the rest XOR key gates in this example to achieve a particular level of security as opposed to using all MUX key gates.

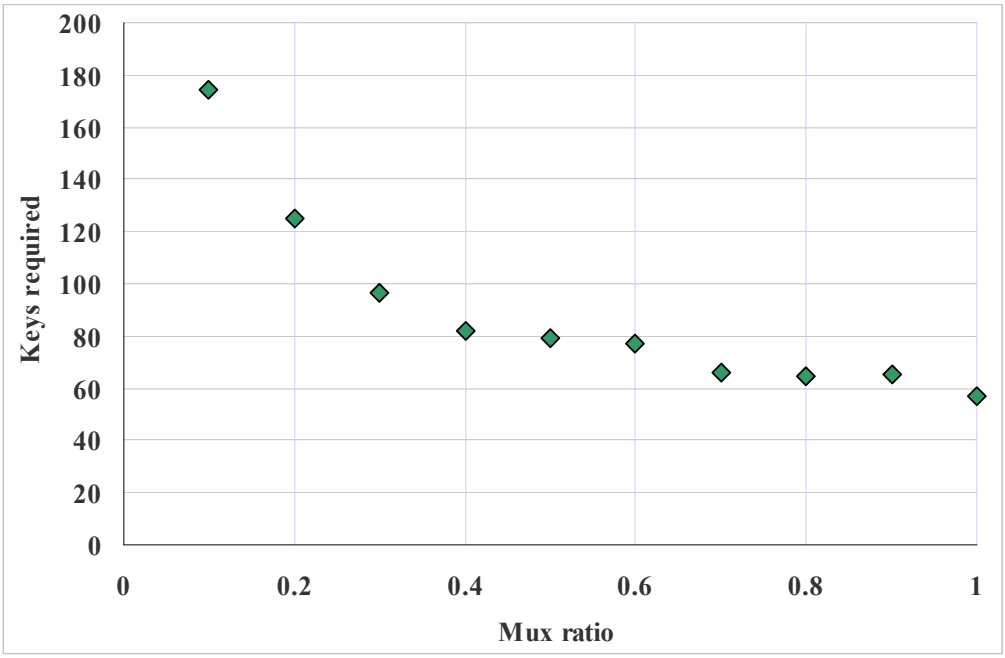


Figure 21. Number of Keys Inserted to Reach 50 Effective Keys for Cone-Based Attack for Circuit *b11*

4.5. CONCLUSIONS

The worst-case number of operations for a brute force attack to determine key values can be significantly lower if each logic cone is iteratively considered going from easiest to hardest. Techniques to counter this are needed to ensure strong logic obfuscation. It was shown that inserting MUX key gates are an effective approach for increasing the security against this type of attack. The most cost effective approach is to use a mix of XOR key gates and MUX key gates.

5. Computing with Obfuscated Data via Noise Insertion and Cancellation

Design-for-trust is an increasingly important topic as more parts of designs are sourced from potentially insecure untrusted IPs. In secure computing, sensitive data must be kept private by protecting it from being obtained by an attacker. The key mechanism for accomplishing this is to encrypt the data. This works well for storing and transferring the data. However, computing with encrypted data is a significant challenge. Existing techniques are either prohibitively expensive (e.g., fully homomorphic encryption [Gentry 09]) or only work for special cases (e.g., only for linear functions [Waksman 11]). Consequently, the conventional approach requires decrypting the data before computing and once the results are obtained, performing encryption at the output (as illustrated in Figure 22). Thus, the raw data is exposed inside the computing unit and effort must be made to keep an attacker from obtaining it either directly or through side-channel attacks.

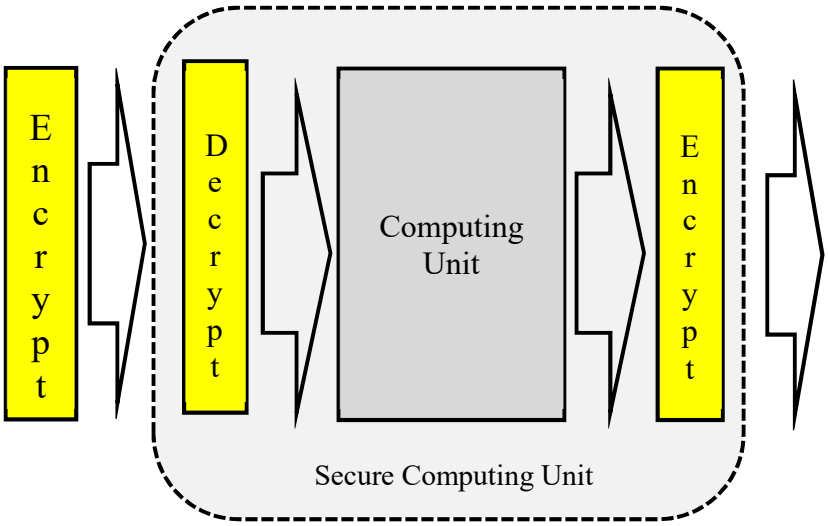


Figure 22. Conventional Secure Computing Scheme

The problem of protecting raw data inside a computing unit is especially challenging for IP modules obtained from a third party. Not only does the third party IP provider know the functionality of the IP module and is aware of potential ways to access the data, but could even maliciously insert a hidden backdoor access mechanism or include a hardware Trojan in the design. Moreover, for a poorly designed third party IP, vulnerabilities in the IP could even be discovered after the hardware is manufactured. Once discovered, these vulnerabilities could be used by any attacker. For example, test mode in a chip might be improperly activated and used to access internal data through scan chains.

The proposed idea for protecting data inside the computing unit is to use a data obfuscation technique that involves inserting noise in the input data and canceling it out both partially internally in the computing unit as well as fully at the output of the computing unit. The proposed scheme is a general technique that can be used for any arbitrary logic circuit. While most conventional secure computing scheme focused on architectural level, the proposed work focuses on gate level security threats. Moreover, it is lightweight and can easily tradeoff the level of data obfuscation with the amount of overhead added to the design. One application of this approach is to protect data being used in third-party IP modules. If the third-party IP is provided as an RTL design, the proposed noise insertion and cancellation logic can be added to the RTL design during synthesis. Since the third-party IP provider does not know the specific design of the noise insertion and cancellation logic, attacks that use backdoor access or hardware Trojan insertion will only yield the obfuscated data. Moreover, computing with obfuscated data may prevent a particular internal transition from triggering a Trojan [Waksman 11]. While the proposed approach does not provide the level of strong encryption that fully homomorphic encryption [Gentry 09] would provide, it has the advantage of being lightweight, easy to implement, and can be deployed with minimal performance impact.

Note that a key idea in the proposed work is to reduce the complexity of the noise cancellation logic by carefully selecting internal locations to do local noise canceling. This is done in a way that prevents more than one input per gate from propagating noise thereby

avoiding the complexity that arises from reconvergent noise propagation paths. By so doing, the proposed methodology is able to significantly reduce the size of the noise cancellation logic required for a user specified level of data obfuscation.

The chapter is organized as follows: Section 5.1. describes related work. Section 5.2. gives an overview of the problem. Section 5.3. explains how to cancel noise within arbitrary computations. Section 5.4. describes a noise path selection algorithm that allows efficient insertion of noise cancelling gates. Section 5.5 presents the experimental results. Section 5.6 discusses an attacker's scenario. Section 5.7 is a conclusion.

5.1. RELATED WORK

Software techniques for general computing with encrypted data based on fully homomorphic encryption [Gentry 09] or garbled circuits [Malkhi 04] exist, but are very computationally expensive and slow. Existing secure computing architectures in hardware (e.g., [Suh 03], [Fletcher 12], [Breuer 13], etc.) are based on decrypting before the computation and encrypting after the computation. Under such schemes, the internal computation is performed without any encryption. Computation on obfuscated data based on noise insertion has been proposed for non-computation logic (which does not alter the data) and linear circuits [Waksman 11]. For logic that does not alter the data, noise is added on one end and removed directly on the other end. For logic that is computing with the data, linear circuits (e.g., an RSA circuit) allow noise to be added and easily removed. A subset of input bits can be flipped with XORs to insert noise to hide the raw data, and it is very easy to cancel the noise at the outputs in a linear circuit because each output depends on the XOR of a subset of input bits. If the number of inputs (with noise inserted) that a particular output depends on is odd, then the output bit is flipped to cancel the noise. In this paper, this concept of inserting and canceling noise is applied for non-linear circuits. In non-linear circuits, canceling out the noise is more complicated and can result in large overhead if it is not implemented in an intelligent way. A design procedure is presented here that partially cancels noise internally in the circuit to keep overhead down.

One of the motivations behind computing with obfuscated data is that third party IPs may not be fully trusted to be free of internal hardware Trojans. Trojans may be inserted at different steps in the logic design flow. At the RTL stage, data obfuscation helps to remove Trojan triggers. At this stage, the third party IP provider is the untrusted party. [Rajendran 15] proposed ways to detect corruption of registers within third party IPs. Trojan detection techniques at RTL level includes but is not limited to behavioral analysis [Salamani 13], functional analysis [Waksman 13], and verification techniques [Zhang 15]. Trojan insertion after synthesis is based on the assumption that the foundry is the untrusted party. Under this assumption, [Wu 16] proposed Trojan prevention and detection based on a randomized parity code and memory based switchboxes. At this level, pure detection techniques based on side channel analysis [Agrawal 07] and built in authentication with signatures [Xiao 14] have been proposed. Security threats from test instruments under a third party IP plug-and-play scenario have been discussed in [Baranowski 13] and [Dworak 13].

5.2. PROBLEM DEFINITION

This section formally defines the problem this chapter is addressing and describes how it can be used to provide security in third party IPs.

Given a computation module M which computes output O with data input I, find an extended module M_x where M is a subset of M_x. M_x generates the same output O with data input I_x which was obfuscated with random key input K through a single level of XOR gates. In other words, the objective is to find M_x which satisfies $M(I) = O = M_x(I_x)$ where $I_x = I$ xor K.

Figure 23 illustrates the overall framework of the proposed scheme. M_x has the same number of outputs as M and has additional key inputs K on top of all the inputs of M. To obfuscate the input I_x to M_x, the XOR cipher C is a module where all inputs of M are XORed with the key K. When M_x is placed on top of the XOR cipher C, and C and M_x are

connected to the same key *K*, the combined module should be functionally equivalent to the original module *M*.

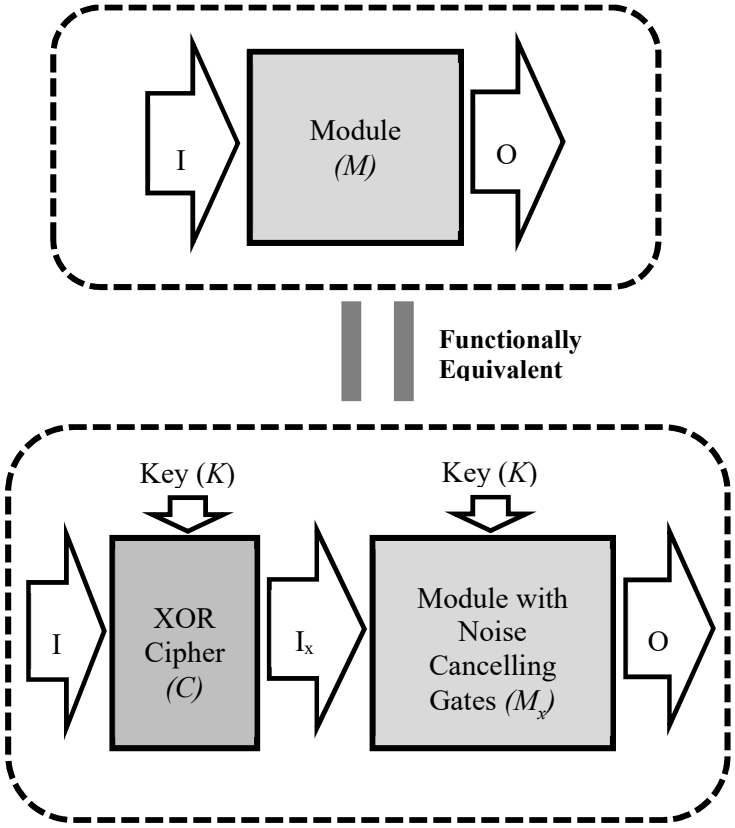


Figure 23. Overview of the Proposed Scheme

Encryption with a XOR cipher can be seen as a process to add random noise to obfuscate the plaintext. This is a common practice for data obfuscation in image processing and communications [Stallings 16]. Conversely, de-obfuscating the signal which contains noise that was previously injected can be seen as a procedure to add noise cancellation logic which effectively cancels the inserted noise that is propagating within the module. In the following sections, the concept of noise cancellation is interchangeable to de-obfuscating the data. The key value *K* is also called the noise source of primary inputs to M_x.

One application of the proposed scheme is for providing security in third-party IPs. In this attacker's scenario, the attacker is either the third party IP provider or whoever knows the vulnerability in the soft IP core (e.g. Verilog code). The attacker does not have access to the netlists sent to a trusted foundry for manufacturing. When the designer gets a soft IP core from a third party, the proposed procedure can be used to add circuitry to insert and cancel noise. The noise can be propagated to particular points inside the module (e.g., particular flip-flops) or simply all the way to the primary outputs. The resulting design M_x is what will be synthesized to generate the hardware. The inputs going into M_x are obfuscated with a XOR cipher and the output can either come out de-obfuscated or obfuscated and later de-obfuscated in a separate module elsewhere. Key values are stored in a secured memory element such as a physically un-clonable function (PUF) [Suh 07] inside M_x or transferred through a secure channel so side-channel attacks on the key values are infeasible. If needed, to strengthen against brute force attacks, the contents of K can be continuously generated with a linear feedback shift register (LFSR). With an LFSR constantly refreshing the key value, the obfuscated value is constantly changing which makes brute force attacks much less effective. The correctness of the computation is not affected as long as both ends of the computation are synchronized to use the same key value.

In sequential design, the noise can propagate to the pseudo primary outputs (at the inputs to the flip-flops). The state flip-flops will then store the obfuscated values. The noise cancelling logic is placed right after the state flip-flops to de-obfuscate the pseudo outputs. If needed, the de-obfuscated value can be obfuscated again by inserting an XOR cipher after the noise cancelling logic. Figure 24 shows the noise cancelling scheme for sequential circuits. In Figure 24, the noise cancellation logic stores the noise cancelling signals to be applied at the next cycle as the values exit the state flip-flops. The noise re-insertion logic obfuscates the pseudo inputs with the current key.

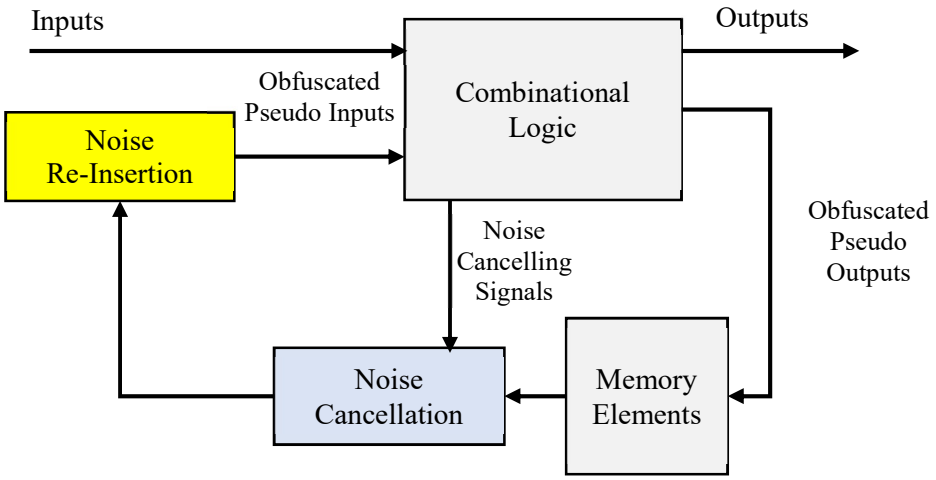


Figure 24. Noise Cancelling Scheme for Sequential Logics

5.3. CANCELLING NOISE IN ARBITRARY COMPUTATIONS

This section describes the proposed approach for designing the noise cancelling circuitry to cancel out the noise injected at the primary inputs (PI) for arbitrary non-linear computational units.

A "noisy fanin" of a gate is defined as a fanin whose value is affected by a noise bit k where k was previously XORed with one of the primary inputs that it depends on, and the effect has not been cancelled. Any fanin that is not noisy is referred to as a "clean fanin". If a 2-input gate has only one noisy fanin and one clean fanin, whether or not the noise propagates to the output depends on whether the clean fanin has a non-controlling value. When the noise does propagate to the output, the noise can be cancelled out by XORing the output with the noise source *k*.

Figure 25 illustrates the gates needed to generate noise cancelling signals for a single AND/OR gate. The picture on the left shows the original gate with input *a* and *b*. On the right input *b* is previously XORed with the noise source and becomes b_k. The path the noise propagates is shown in bold. To cancel the noise introduced by b_k., one AND gate

is created to check the condition necessary to propagate the noise to *F*. This AND gate can then be used to cancel noise through XORing the output of the new AND gate with the output of the original gate with the noisy fanin.

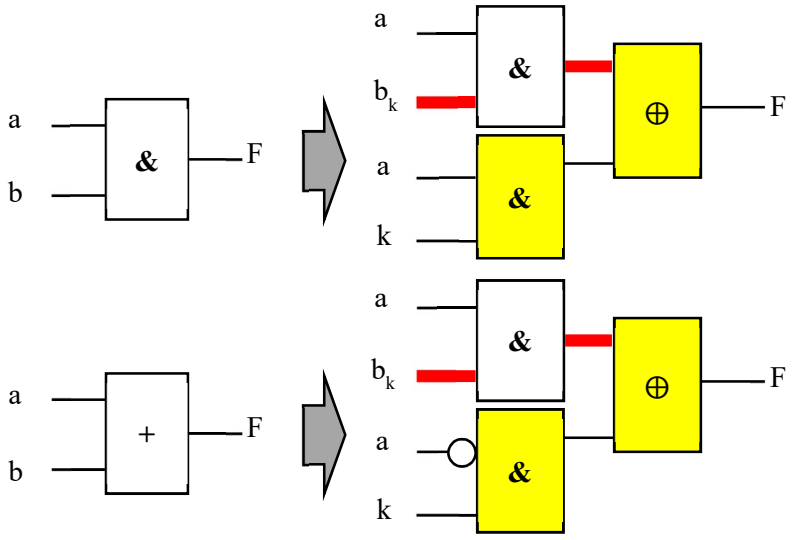


Figure 25. Noise Cancelling Gates for AND/OR

When the noise further propagates through more than one gate, the gates it has traversed through form a "noise propagation path" or simply called "noise path." The noise can be cancelled out by checking the noise sensitization conditions along the noise path. When every gate on the noise path has only one noisy fanin, the noise sensitization condition depends only on the clean fanins along the path. In this case, the sensitization condition logic basically repeats the computation along the path by retaining all the clean fanins and replacing the noisy fanin with the propagating noise. When both fanins are noisy, the process first generates the noise cancelling XOR for one of the noisy fanins to clean it before it feeds the gate. This ensures that there is never more than one noisy fanin for any gate on a noise path.

Figure 26 shows an example where the noise further propagates and the gates that are needed to remove the noise are shown. The top left picture shows the original module which contains nine gates, three PIs and three POs. On the bottom of the original module is an example of a XOR cipher which produces obfuscated inputs, which can be considered the noise source. *K* is the key stored in secured memory elements. The inputs to the original module are XORed with the key values and thus *a* becomes a_k., *b* become b_k. and *c* become c_k.. The inputs obfuscated by the XOR cipher are fed to the module with noise cancelling gates inserted. The module on the right is the final module with noise cancelling gates inserted. In this example, the deepest point to which the noise propagates are the primary outputs *O1* and *O3*. The bold path shows where the noise propagates in the module. For simplicity, most of the noise is cancelled at the first level and PI *b* is already noise-free. The primary input that is actually noisy is marked above the XOR used for cancelling noise.

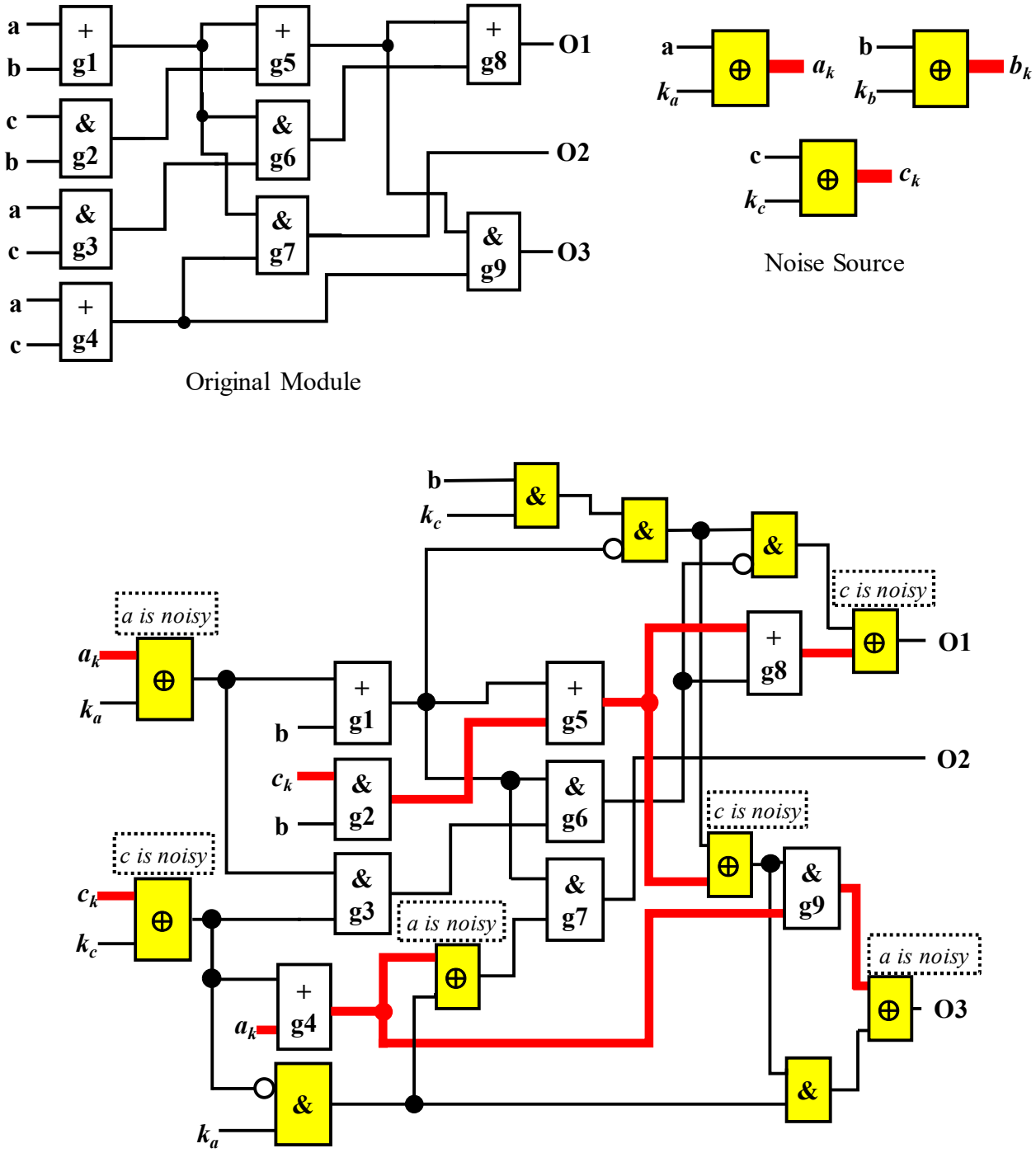


Figure 26. Example of Inserting Noise Cancelling Gates with Noise Paths Shown in Bold

5.4. NOISE PROPAGATION PATH SELECTION IN AIGS

The procedure described in Section 5.3 cancels noise propagating in a module with XOR and AND gates. The noise path needs to be chosen carefully to keep overhead from the additional XOR and AND gates down. This section describes the proposed algorithm based on and-inverter graphs (AIG) that is used to select noise propagation paths and then determine how many XOR and AND gates are needed to cancel the noise.

An AIG is a directed and acyclic graph that represents the structure of a circuit. Figure 27 presents an AIG of the same module shown in Figure 26. In Figure 27, Shaded nodes represents PO and nodes with dotted outline represent PI. Each node in the AIG has two inputs, representing their logical conjunction. The edges between the nodes may contain markers which indicate logical negation. AIGs are widely used in logic synthesis because of their simple structure which allows efficient manipulation of large logic networks [Kuehlmann 02].

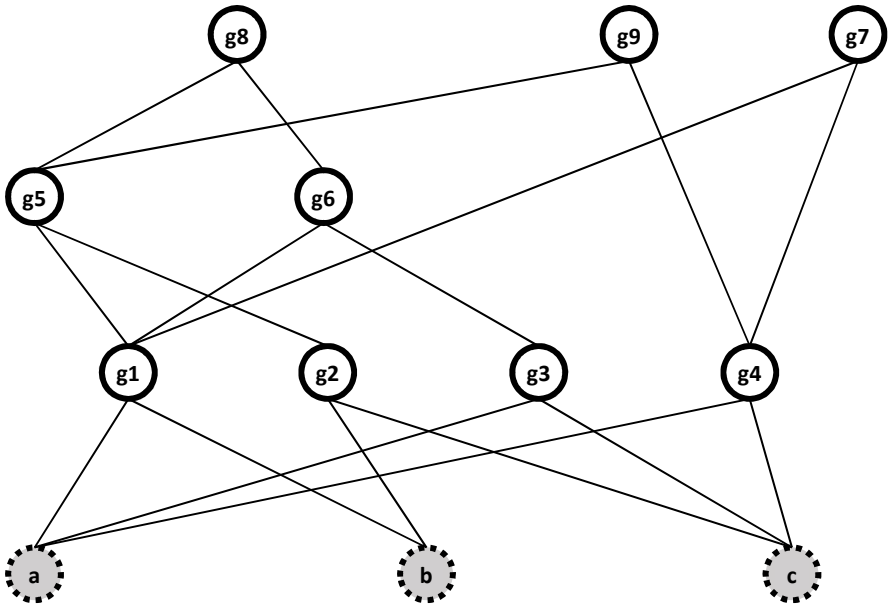


Figure 27. AIG of the same module shown in Figure 26.

The inputs to the path selection algorithm are the module and the targets where the noise needs to be propagated to. The noise paths are constructed from the target nodes, one path at a time. First, the entire module is represented as an AIG. The heuristic then performs a search starting from the targets within the AIG in a depth first search fashion. Whenever a node is traversed, its fanins are examined to determine whether it is necessary to first traverse to its fanins before putting it into the noise path. The search always stops upon hitting a PI. Since the goal of the heuristic is to encourage sharing in order to lower the number of additional noise cancelling gates needed, the algorithm always picks the fanin that is either already on a noise path or the one closest to a PI.

After paths are selected, each AIG node will be labeled with different noise levels. The noise level can be either clean, mixed, or full. Clean nodes are not on any noise path selected. Mixed nodes are on at least one noise path but not all of its fanouts are on the same noise path as this node. Full nodes are on at least one noise path and all of its fanouts are on a noise path that includes this node. The difference between a full and a mixed node is that a full node does not need a noise cancelling XOR for any of its fanouts. For each mixed node, one AND gate and one XOR gate are required, where the XOR gate becomes the new fanin of all fanouts that is either clean or belong to the same noise path as this node. The AND gate becomes the new fanin for all fanouts on the same noise path as this node. For each full node, an AND gate is created and becomes the new fanin for all of its fanouts.

Figure 28 presents an example of noise path selection on the AIG of the original module in Figure 26. For simplicity, inversion markers are not shown in the example. In this example, the process begins from each of the targets, which in the example are the POs. Since sharing nodes on the path helps to reduce the number of noise cancelling nodes, the algorithm always chooses a node already used by another path. In this example, there are three paths starting from *g7*, *g8* and *g9* respectively. Node *g5* is an example of a full node. Node *g1* is an example of a mixed node. Overall, six XOR gates are needed: three for cancelling noise at *g7*, *g8* and *g9* because they are POs; three for cancelling noise at

input *a, b* and *c* because they are either clean or mixed nodes. Five AND gates are needed to compute the noise sensitization condition.

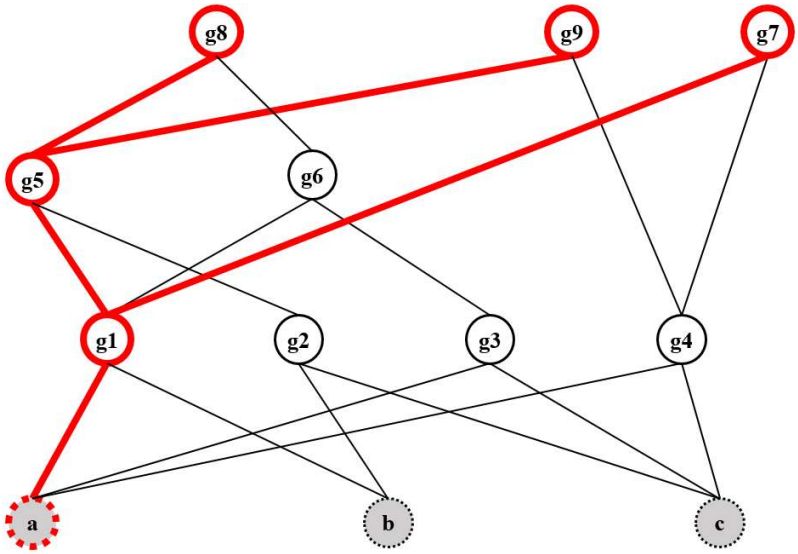


Figure 28. Example of Selecting Noise Paths in AIG for Module in Figure 26.

5.5. EXPERIMENTAL RESULTS

All aforementioned algorithms have been implemented and experiments were performed on the ISCAS and ITC benchmark circuits [Brglez 85, Corno 00]. Construction and manipulation of AIGs was done using the AIG package provided by ABC [Brayton 10].

Results are shown in Table 9 where the noise targets used are all of the POs. The table shows the circuit name followed by the number of PIs and POs that is has and the number of gate equivalents (GEs). GEs are computed assuming 1 GE per two-input AND/OR gate with an XOR being equivalent to 1.5 GE [Martins 15]. The remainder of the table shows the results after the noise cancelling gates have been inserted. The number

of AND and XOR gates is shown followed by the corresponding gate equivalents. Overhead is shown with respect to the original circuit paired with a XOR decipher. The last column shows the “noise ratio” which is the percentage of internal gates that have a noisy fanin. Even though the proposed noise cancelling gates partially repeat the computation up until the POs to cancel noise, the results show that the overhead of adding noise cancelling is much lower than duplicating the entire computation. This is because the noise path selection algorithm encourages sharing of noise cancelling gates between different noise propagation paths. Regardless of the number of POs or the depth of logic, the more logic sharing between the primary outputs, the lower the overhead. As a result, the proposed noise path selection algorithm propagate noise to the POs with a relatively low noise ratio since the algorithm always tries to share nodes between paths.

Table 9. Results for Inserting Noise Cancelling Gates

Benchmark	Original Module with XOR Decipher			Module with Noise Cancelling Gates				
				AIG		Combined Network		
	#PI	#PO	GE	#XOR	#AND	GE	Overhead	Noise Ratio
C432	36	7	190	48	21	283	16%	11%
C499	41	32	392	73	64	566	25%	16%
C880	60	26	314	85	48	490	21%	15%
C1355	41	32	504	141	168	884	56%	39%
C1908	33	25	414	60	47	551	19%	12%
C2670	233	139	717	298	77	1241	16%	13%
C3540	50	22	1038	96	76	1258	13%	7%
C5315	178	123	1773	313	192	2435	19%	11%
C6288	32	32	2337	527	524	3652	53%	22%
C7552	207	108	2074	372	248	2880	21%	16%
b14	32	54	6115	702	590	7758	26%	10%
b15	36	70	8454	1437	996	11606	36%	12%
b17	37	97	27581	4598	3238	37716	36%	12%
b20	32	22	12189	1453	1182	15551	27%	10%
b21	32	22	12746	1514	1167	16184	26%	9%
b22	32	22	18453	2250	1729	23557	27%	9%
b14	32	54	6115	702	590	7758	26%	10%

In Table 9, the number of noise paths selected is the same as the number of POs in the module. It is possible to select more noise propagation paths by adding more nodes to the set of noise targets. Doing so increases the noise ratio but also requires more overhead. Figure 29 shows the tradeoff between increasing the noise ratio versus the overhead of the corresponding noise cancelling network for three benchmark circuits. After the proposed algorithm has selected the noise paths for each POs, additional non-noisy nodes are selected to increase the noise ratio. The noise ratio in Figure 29 starts from 30% since in most cases the noise ratio does not exceed 30% after noise paths are selected for all POs. As can be seen in Figure 29, the amount of overhead grows nearly linearly as the noise ratio is increased. In practice, after noise propagated to all necessary targets, designer can set an arbitrary noise ratio target and select more noise paths in the module.

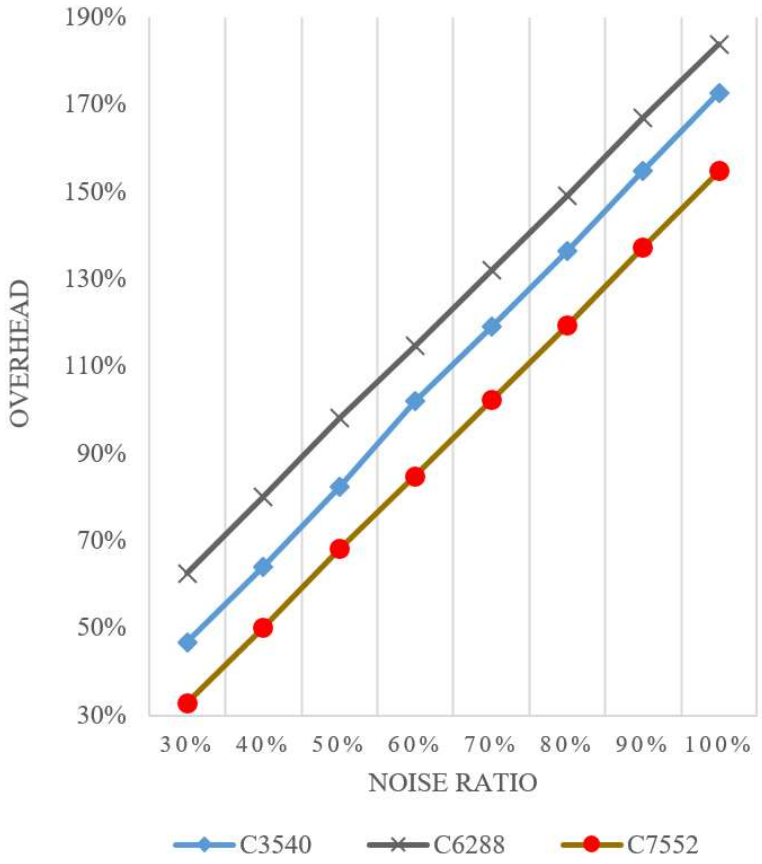


Figure 29. Overhead vs. Noise Ratio

5.6. ANALYSIS

This section describes a possible attacker's scenario and how the proposed approach can help to defend against it.

As discussed in the literature, Trojans inserted at the RTL level are significant security threat, especially when the design may be reused in a SoC environment where the source code may come from different sources [Waksman 13]. Various approaches have been proposed to detect them, but a carefully designed Trojan may still go undetected and get synthesized into the final hardware. Even if a Trojan is not inserted, the attacker may have knowledge about an existing security leak present in the RTL code. The effect of

having a security leak in the untrusted module includes the case where an attacker may have unauthorized control over the module, which gives an attacker not only the ability to observe the internal activity of the module, but also the ability to retrieve data generated from a trusted module that communicates directly with the untrusted module. For example, the boundary scan in the unsecured module can be a gateway for a malicious user to observe the output from a trusted module that the attacker wants to observe. In traditional secure computing, data is de-obfuscated before the computation, and the existence of scan cells in the untrusted module allows any malicious user to retrieve the plaintext computed by the trusted module that the untrusted module is communicating with. With the proposed scheme, the data during the computation is obfuscated even in the untrusted module. To retrieve the plaintext, an attacker needs to have knowledge about the obfuscation technique and key being used.

The proposed obfuscation scheme with a stream cipher is based on XOR ciphers used in conjunction with a one-time pad (OTP). The security of a OTP based obfuscation depends on how the key is generated and managed [Zeng 91]. In the proposed scenario, key values can be stored in a tamper proof LFSR. An attacker may attempt to observe the value stored in the LFSR by waiting for the pattern of LFSR to repeat after a limited amount of clock cycles. Irregular clocking [Gollman 89] can be employed to mitigate this kind of attack. The design and the LFSR used for obfuscation can be paired with different clock source. In particular, the LFSR clock can be randomly disturbed so the attacker can not easily find out when the LFSR will starting repeating.

Note that the proposed approach is not intended for prevention or detection of hardware Trojans. The motivation behind computing with obfuscated data is to prevent plain data from being retrieved easily should a security leak be present in an untrusted design. The security leak can be due either to a well-crafted Trojan or to a poorly designed module.

5.7. CONCLUSIONS

The proposed methodology for computing with obfuscated data is lightweight and can be applied for any arbitrary logic circuit. One important application is to prevent vulnerabilities in a soft IP core from being exploited to gain access to sensitive data during computation. While most secure computing schemes are focused at the architectural level, the proposed work is implemented at the gate level. Results show that the overhead is much lower than duplicating the entire computation to propagate noise to all outputs with maximal sharing of noise cancelling gates between noise paths. The noise ratio can be easily increased at the cost of additional overhead by adding more noise propagation targets and thus creating more noise paths in the module.

6. Summary and Future Works

This chapter summarizes the contributions of this dissertation and suggests areas for future work.

Chapter 2 describes an approach for designing a single TAM architecture with a "bandwidth adapter" on each die that can be used efficiently for multiple test data bandwidths. Experimental results are presented which shows that this approach allows efficient test in all phases from pre-bond, multiple partial stack configurations, and post-bond.

Chapter 3 presents an improvement over conventional sequential linear decompression. It uses a multiple polynomial LFSR with a counter to rotate between different polynomials which eliminates the need for control bits to select the polynomial. It can encode more test cubes than can be encoded with a fixed polynomial. Results show improvement in number of encodable test cubes as well as the number of free variables retained when being used with a FIFO. This results in an improvement in overall compression achieved.

Chapter 4 describes a new attack strategy against logic obfuscation. It is based on applying brute force iteratively to each logic cone and is shown to significantly reduce the number of brute force key combinations that need to be tried by an attacker. Different heuristics for key gate insertion against the new attack strategy are proposed and discussed. It is shown that inserting key gates based on MUXes is an effective approach to increase security against this type of attack. Experimental results are presented quantifying the threat posed by this type of attack along with the relative effectiveness of MUX key gates in countering it.

Finally, Chapter 5 presents a computation scheme with obfuscated data. Conventional secure computing needs decryption before the actual computation is performed which exposes the data to security leaks within the computing unit. The proposed scheme avoids this by performing computation on noise-obfuscated data. The proposed scheme is lightweight and can be applied for any arbitrary logic circuit. One important application is

to prevent vulnerabilities in third party IPs from being exploited to gain access to sensitive data during computation.

There are several directions for future work. The presented work in secure computing is based on the concept that data is obfuscated with a key which will be required for de-obfuscation later. The computation is based on hardware redundancy which removes noise wherever necessary. Alternatively, time redundancy could be utilized for secure computing. One computation can be split into different computations, where each is obfuscated differently and the final results can only be retrieved by combining separate computations together. In combinational circuits, some input bits can be obfuscated by noise. When evaluating the result, only the logic cone not affected by noise is evaluated. The key is not required for obtaining the final results when the obfuscated bit location is known. While the idea is straightforward for combinational circuits, how to apply this concept to sequential circuits is a challenging task worthy of further research. In addition, the proposed work in Chapter 5 is the very first step to bridging the fully homomorphic encryption from the software domain to the hardware domain. The proposed work focused on using XORs and one time pad (OTP) as obfuscation hardware. While it has the advantage of being lightweight, the level of security can be much improved with more powerful encryption algorithms. It is worthy of further research to see if computing with obfuscated data is possible in techniques other than XOR with an OTP.

References

[Agrawal 07] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi and B. Sunar, "Trojan Detection using IC Fingerprinting," *Proceedings of IEEE Symposium on Security and Privacy*, pp. 296-310, 2007.

[Baranowski 13] R. Baranowski, M. A. Kochte and H. J. Wunderlich, "Securing Access to Reconfigurable Scan Networks," *Proceedings of Asian Test Symposium*, pp. 295-300, 2013.

[Baumgarten 10] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC Piracy Using Reconfigurable Logic Barriers," *IEEE Design and Test of Computers*, Vol. 27, No. 1, pp. 66-75, Jan. 2010.

[Brayton 10] R. Brayton and A. Mishchenko. "ABC: an academic industrial-strength verification tool," *Proceedings of International Conference on Computer Aided Verification*, pp. 24-40, 2010.

[Breuer 13] P.T. Breuer and J.P. Bowen, "A fully homomorphic crypto-processor design," *Proceedings of International Symposium on Engineering Secure Software and Systems*, pp. 123-138, 2013.

[Brglez 85] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits," *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 695-698, 1985.

[Chakraborty 09a] R. Chakraborty and S. Bhunia, "Security against hardware Trojan through a novel application of design obfuscation," *Proceedings of International Conference on Computer-Aided Design*, pp.113-116, 2009.

[Chakraborty 09b] R. Chakraborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Transactions on Computer-Aided Design*, Vol. 28, No. 10, pp. 1493-1502, Oct. 2009.

[Corno 00] F. Corno, M. S. Reorda and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Design and Test of Computers*, vol. 17, no. 3, pp. 44-53, Jul/Sep 2000.

[Dworak 13] J. Dworak, A. Crouch, J. Potter, A. Zygmontowicz and M. Thornton, "Don't forget to lock your SIB: hiding instruments using P1687," *Proceedings of IEEE International Test Conference*, pp. 1-10, 2013.

[Fletcher 12] C.W. Fletcher, M. Dijk, and S. Devadas. "A secure processor architecture for encrypted computation on untrusted programs," *Proceedings of ACM workshop on Scalable trusted computing*, pp. 3-8, 2012.

[Gentry 09] C. Gentry, "Fully homomorphic encryption using ideal lattices," *Proceedings of ACM symposium on Theory of computing,* 2009.

[Goel 03] S.K. Goel, and E.J. Marinissen, "Effective and Efficient Test Architecture Design for SOCs," *Proceedings of International Test Conference*, pp. 529-538, 2002.

[Gollmann 89] D. Gollmann and W. G. Chambers, "Clock-controlled shift registers: a review," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 4, pp. 525-533, May 1989.

[Goppa 70] V.D. Goppa, "A new class of linear correcting codes," *Problemy Peredachi Informatsii,* Vol. 6.3, pp. 24-30, 1970.

[Hellebrand 95] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataramann, and B. Courtois, "Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *IEEE Transactions on Computers*, Vol. 44, Iss. 2, pp. 223-233, Feb. 1995.

[IEEE 1149.1-2001] IEEE Std. 1149.1-2001, "IEEE Standard Test Access Port and Boundary-Scan Architecture," *IEEE Press*, 2001.

[Inoue 09] M. Inoue et al. "Partial scan approach for secret information protection," *Proceedings of IEEE International Test Symposium*, 2009.

[Iyengar 02] V. Iyengar, K. Chakrabarty, and E.J. Marinissen, "Test Wrapper and Test Access Mechanism Co-Optimization for System-on-a-Chip," *Journal of Electronic Test: Theory and Applications*, Vol. 18, p. 213-230, Apr. 2002.

[Jiang 09] L. Jiang, L. Huang, and Q. Xu, "Test Architecture Design and Optimization for Three-Dimensional SoCs," *Proceedings of Design Automation and Test in Europe*, 2009.

[Jiang 12] L. Jiang, Q. Xu, K. Chakrabarty, and T.M. Mak, "Integrated Test-Architecture Optimization and Thermal-Aware Test Scheduling for 3-D SoCs Under Pre-Bond Test-Pin-Count Constraint," *IEEE Transactions on VLSI*, Vol. 20, No. 9, pp. 1621-1633, Sep. 2012.

[Keim 13] M. Keim, "Thinking About Adopting IEEE P1687?," *IEEE Design and Test*, vol. 30, no. 5, pp. 36-43, Oct. 2013.

[Khoche 01] A. Khoche, R. Kapur, D. Armstrong, T.W. Williams, M. Tegethoff, and J. Rivoir, "A new methodology for improved tester utilization," *Proceedings of IEEE International Test Conference*, pp. 916-923, 2001.

[Könemann 91] B. Könemann, "LFSR-Coded Test Patterns for Scan Designs," *Proceedings of European Test Conference*, pp. 237-242, 1991.

[Könemann 01] B. Könemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, "A SmartBIST Variant with Guaranteed Encoding," *Proceedings of Asian Test Symposium*, pp. 325-330, 2001.

[Koranne 03] S.Koranne, "Design of Reconfigurable Access Wrappers for Embedded Core Based SoC Test," *IEEE Transactions on VLSI Systems*, Vol. 11, No. 5, pp. 955-960, Oct. 2003.

[Krishna 01] C.V. Krishna, A. Jas, and N.A. Touba, "Test Vector Encoding Using Partial LFSR Reseeding," *Proceedings of International Test Conference*, pp. 885-893, 2001.

[Kuehlmann 02] A. Kuehlmann, V. Paruthi, F. Krohm and M. K. Ganai, "Robust Boolean reasoning for equivalence checking and functional property verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1377-1394, Dec 2002.

[Larsson 05] E. Larsson, "Introduction to Advanced System-on-Chip Test Design and Optimization," *Springer*, 2005.

[Lee 06] J. Lee, M. Tebranipoor, and J. Plusquellic, "A low-cost solution for protecting IPs against scan-based side-channel attacks," *Proceedings of VLSI Test Symposium*, 2006.

[Lee 13] Y.-W. Lee and N.A. Touba, "Unified 3D Test Architecture for Variable Test Data Bandwidth Across Pre-Bond, Partial Stack, and Post-Bond Test," *Proceedings of IEEE Symposium on Defect and Fault Tolerance,* Paper 7.5, 2013.

[Lee 15] Y.-W. Lee and N.A. Touba, "Improving Logic Obfuscation via Logic Cone Analysis," *Proceedings of IEEE Latin-American Test Symposium*, 2015.

[Malkhi 04] D. Malkhi, N. Noam, P. Benny, and S. Yaron, "Fairplay-Secure Two-Party Computation System," *Proceedings of the 13th USENIX Security Symposium*, 2013.

[Marinissen 99] E.J. Marinissen et al, "Towards a standard for embedded core test: An example," *Proceedings of International Test Conference*, 1999.

[Marinissen 00] E.J. Marinissen, S.K. Goel, and M. Lousberg, "Wrapper design for embedded core test," *Proceedings of International Test Conference*, pp. 911–920, 2000.

[Marinissen 02] E.J. Marinissen, V. Iyengar, and K. Chakrabarty, "A set of benchmarks for modular testing of SOCs," *Proceedings of International Test Conference*, pp. 519–528, 2002.

[Marinissen 09] E.J. Marinissen, and Y. Zorian, "Testing 3D Chips Containing Through-Silicon Vias," *Proceedings of International Test Conference*, paper ET 1.1, 2009.

[Martins 15] M. Martins, J.M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen, "Open Cell Library in 15nm FreePDK Technology," *Proceedings of 2015 Symposium on International Symposium on Physical Design*, pp. 171-178, 2015.

[Muthyala 12] S. S. Muthyala and N. A. Touba, "Improving test compression by retaining non-pivot free variables in sequential linear decompressors," *Proceedings of International Test Conference*, pp. 1-7, 2012.

[NCSU 11] NCSU EDA FreePDK45 Wiki, Last modified 2011, http://www.eda.ncsu.edu/wiki/FreePDK45:Contents

[Noia 10a] B. Noia, S.K. Goel, K. Chakrabarty, E.J. Marinissen, and J. Verbree, "Test-Architecture Optimizations for TSV-Based 3D Stacked ICs," *Proceedings of European Test Symposium*, pp. 24-29, 2010.

[Noia 10b] B. Noia, K. Chakrabarty, and E.J. Marinissen, "Optimization Methods for Post-Bond Die-Internal/External Testing in 3D Stacked ICs," *Proceedings of International Test Conference*, Paper 6.3, 2010.

[Noia 11] B. Noia, K. Chakrabarty, S.K. Goel, and E.J. Marinissen, "Test-Architecture Optimization and Test Scheduling for TSV-Based 3-D Stacked ICs", *IEEE Transactions on Computer-Aided Design*, Vol. 30, No. 11, Nov. 2011.

[Pless 11] V. Pless, "Introduction to the theory of error-correcting codes," *John Wiley & Sons*, 3rd Edition, pp. 58-59, 2011.

[Rajendran 12] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security Analysis of Logic Obfuscation," *Proceedings of Design Automation Conference,* pp. 83-89, 2012.

[Rajendran 14] J. Rajendran, H. Zhang, C. Zhang, G. Rose, Y. Pino, O. Sinanoglu, R. Karri, "Fault Analysis-Based Logic Encryption," *IEEE Transactions on Computers*, 2014.

[Rajendran 15] J. Rajendran, V. Vedula and R. Karri, "Detecting malicious modifications of data in third-party intellectual property cores," *Proceedings of Design Automation Conference*, pp. 1-6, 2015.

[Rajski 04] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded Deterministic Test," *IEEE Transactions on Computer-Aided Design*, Vol. 23, Issue 5, pp. 1306-1320, May 2004.

[Rosenfeld 10] K. Rosenfeld, and R. Karri, "Attacks and Defenses for JTAG," *IEEE Design and Test of Computers, vol.* 27.1, pp. 36-47, 2010.

[Roy 08] J. Roy, F. Koushanfar, and I. Markov, "EPIC: Ending Piracy of Integrated Circuits," *Proceedings of Design Automation and Test in Europe*, pp. 1069-1074, 2008.

[Roy 10] J. Roy, F. Koushanfar, and I. Markov, "Ending Piracy of Integrated Circuits," *IEEE Computer*, pp. 30-38, 2010.

[Salamani 13] H. Salmani and M. Tehranipoor, "Analyzing circuit vulnerability to hardware Trojan insertion at the behavioral level," *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems,* pp. 190-195, 2013.

[Stallings 16] W. Stallings, "Cryptography and Network Security: Principles and Practices*," Pearson/Prentice Hall*, 2016.

[Suh 03] G. E. Suh, D. Clarke, B. Gassend, M. Dijk, and S. Devadas, "AEGIS: architecture for tamper-evident and tamper-resistant processing," *Proceedings of ACM International Conference on Supercomputing*, pp. 357-368, 2003.

[Suh 07] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," *Proceedings of ACM/IEEE Design Automation Conference*, pp. 9-14, 2007.

[Taouil 10] M. Taouil, S. Hamdioui, K. Beenakker, and E.J. Marinissen, "Test Cost Analysis for 3D Die-to-Wafer Stacking," *Proceedings of Asian Test Symposium*, pp. 435-441, 2010.

[Touba 06] N.A. Touba, "Survey of Test Vector Compression Techniques," *IEEE Design and Test Magazine*, Vol. 23, Issue 4, pp. 294-303, Jul. 2006

[Waksman 11] A. Waksman and S. Sethumadhavan, "Silencing Hardware Backdoors," *Proceedings of IEEE Symposium on Security and Privacy*, pp. 49-63, 2011.

[Waksman 13] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: identification of stealthy malicious logic using boolean functional analysis," *Proceedings of ACM SIGSAC conference on Computer and Communications Security*, pp. 697-708, 2013.

[Wu 08] X. Wu, Y. Chen, K. Chakrabarty, and Y. Xie, "Test-access Mechanism Optimization for Core-based Three-Dimensional SOCs," *Proceedings of International Conference on Computer Design*, pp. 212-218, 2008.

[Wu 16] T. F. Wu, K. Ganesan, Y. A. Hu, H. S. P. Wong, S. Wong and S. Mitra, "TPAD: Hardware Trojan Prevention and Detection for Trusted Integrated Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 4, pp. 521-534, April 2016.

[Xiao 14] K. Xiao, D. Forte and M. Tehranipoor, "A Novel Built-In Self-Authentication Technique to Prevent Inserting Hardware Trojans," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1778-1791, Dec. 2014.

[Xu 05] Q. Xu and N. Nicolici, "Resource-Constrained System-on-a-Chip Test: A Survey," *IEEE Computers and Digital Techniques*, vol. 152, iss. 1, pp. 67-81, Jan. 2005.

[Yang 04] B. Yang, K. Wu, and R. Karri, "Scan based side channel attack on dedicated hardware implementations of data encryption standard," *Proceedings of International Test Conference*, 2004.

[Zeng 91] K. Zeng, C. H. Yang, D. Y. Wei and T. R. N. Rao, "Pseudorandom bit generators in stream-cipher cryptography," *IEEE Computer,* vol. 24, no. 2, pp. 8-17, Feb. 1991.

[Zhang 15] J. Zhang, F. Yuan, L. Wei, Y. Liu and Q. Xu, "VeriTrust: Verification for Hardware Trust," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 34, no. 7, pp. 1148-1161, July 2015.

Vita

Yu-Wei Lee received his Bachelor of Science (B.S.) degree in Computer Science from the Department of Computer Science, National Chiao-Tung University (NCTU), Hsinchu, Taiwan in June 2007. In 2009, he received a Master of Science (M.S.) degree from the Department of Computer Science, National Chiao-Tung University, Hsinchu, Taiwan. He began his Ph.D. studies at the University of Texas at Austin in August 2011. His research interests are Design for Test and Trust (DFT), Hardware Security, and Algorithms for Computer Aided Design.

Permanent address (or email): ywlee@utexas.edu

This dissertation was typed by Yu-Wei Lee.