POZNAŃ UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATIONS

**BARTOSZ WŁODARCZAK**

# ON SECURE DETERMINISTIC IN-SYSTEM TEST SOLUTIONS

Ph. D. Thesis

Supervisors:

prof. dr inż. Janusz Rajski
prof. dr hab. inż. Jerzy Tyszer

Poznań, Poland, 2024

# ABSTRACT

In the ever-evolving world of integrated circuits (ICs), manufacturing processes have made it possible to deliver designs of staggering complexity with billions of transistors placed on a single silicon die. However, with the very small feature sizes, these technologies are extremely fragile and vulnerable to new types of failure mechanisms and defects. In the automotive domain, hyperscale data centers, healthcare ICs, and many other applications electronic designs must be continuously tested during a product lifecycle to avoid malfunctions caused by, for example, silicon degradation. A deterministic in-system test is one of the most prominent solutions, capable of detecting defects throughout the lifecycle of state-of-the-art ICs. Although it can significantly improve the in-field test quality, the very same test infrastructure and other DFT schemes may expose a design to many security threats. Clearly, securing the electronic devices that underpin the global economy, businesses and personal lives has become essential in the face of growing cybersecurity threats. In particular, on-chip test instruments have to be protected against unauthorized access and other malicious activities. To satisfy current and anticipated VLSI test requirements, the thesis introduces a number of solutions that target two important aspects of the deterministic in-system test paradigm: advanced test response compaction and in-system test security.

In the first part of the thesis, new X-masking methods devoted to the in-system test response compaction are examined. The first compactor is designed for a logic built-in self-test environment. Furthermore, it is capable of handling test data produced by observation scan chains that may capture errors at every single scan shift cycle. The second solution is strictly integrated with a deterministic in-system test. As a result, this X-masking scheme receives controls from an on-chip test data decompressor. In addition to design principles of selection logic, the rules that govern the encoding of masking data are also discussed.

The subsequent part of the thesis introduces new lightweight cryptographic schemes which when working synergistically, may form a hardware root of trust destined to protect the design's IP and defend test infrastructure against intrusions. This part begins with a hybrid ring generator (HRG), a modified version of a conventional ring generator. Among several HRG applications, the work proposes three new lightweight cryptographic primitives: a crypto hash function, a stream cipher of test data, and a true random number generator. Finally, a hardware root of trust is presented that builds on just described primitives to facilitate development of challenge-response authentication protocols. The solution has a low area footprint, operates at very high frequencies, and is fully compatible with a design and DFT flow. Although it primarily targets SSN-based designs, i.e., System-on-Chip solutions with packetized streaming of test data, the proposed root of trust can improve the security of other test interfaces, as well.

Both test response compactors have been thoroughly examined through experiments conducted on large and complex industrial designs representing the latest technology nodes while varying with respect to design styles and scan methodologies. The new security primitives, on the other hand, have been verified using batteries of statistical tests, including those provided by National Institute of Standards and Technology (NIST) and BSI - the German IT security certification authority.

## STRESZCZENIE

W dynamicznie rozwijającej się domenie scalonych układów cyfrowych, procesy ich wytwarzania umożliwiły dostarczanie układów o bezprecedensowej złożoności z miliardami tranzystorów umieszczonymi na pojedynczej matrycy krzemowej. Niestety, w związku z bardzo małymi rozmiarami elementów półprzewodnikowych, nowe technologie są niezwykle wrażliwe i podatne na nowe rodzaje uszkodzeń. Układy scalone używane w przemyśle motoryzacyjnym, medycynie, w centrach danych oraz w wielu innych zastosowaniach muszą być testowane przez cały okres ich eksploatacji, aby uniknąć nieprawidłowego działania spowodowanego m.in. starzeniem się układu. Deterministyczne testowanie systemowe to jedno z najbardziej obiecujących rozwiązań, pozwalających na wykrywanie uszkodzeń w trakcie eksploatacji najnowszych układów scalonych. Takie podejście umożliwia znaczne podniesienie jakości testowania, jednak może jednocześnie zostać wykorzystane do nielegalnego zidentyfikowania wewnętrznej struktury lub funkcjonalności układu. Zabezpieczenie urządzeń elektronicznych staje się niezbędne, szczególnie w obliczu rosnącej liczby zagrożeń związanej z cyberbezpieczeństwem. Narzędzia testujące w układach scalonych muszą być w szczególności chronione przed nieautoryzowanym dostępem i innymi działaniami o wrogim charakterze. W związku z przedstawionymi wymaganiami w pracy przedstawiono rozwiązania, które koncentrują się na dwóch istotnych aspektach deterministycznego testowania wbudowanego: zaawansowanej kompakcji (redukcji) odpowiedzi testowych oraz bezpieczeństwie narzędzi testujących.

W pierwszej części rozprawy podano metody eliminacji stanów nieznanych dla kompakcji odpowiedzi testowych. Pierwsze rozwiązanie zostało zaprojektowane dla wbudowanego testu, opartego o pseudolosowe wektory testowe. Zaproponowane podejście umożliwia również kompakcję odpowiedzi testowych wygenerowanych przez ścieżki testujące, które pobierają dane o uszkodzeniach w każdym cyklu zegara. Drugie rozwiązanie jest ściśle zintegrowane z wbudowanym w system deterministycznym testowaniem. W rezultacie sterowanie dla nowego kompaktora pochodzi z umieszczonego na chipie dekompresora danych. Poza szczegółami dotyczącymi projektowania układów maskujących stany nieznane, w pracy przedstawione zostały również zasady kodowania danych sterujących kompaktorami.

W drugiej części rozprawy przedstawione zostały nowe techniki kryptograficzne, które mogą stanowić bazę dla sprzętowego rozwiązania zapewniającego obronę infrastruktury testującej przed niepożądanym dostępem. Ta część rozpoczyna się wprowadzeniem hybrydowego generatora pierścieniowego. Wśród kilku zastosowań tego układu, w pracy zaproponowano trzy nowe moduły kryptograficzne: kryptograficzną funkcję skrótu, generator dla potrzeb szyfrowania strumieniowego oraz sprzętowy generator liczb prawdziwie losowych. Następnie zostały one wykorzystane w ostatnim rozdziale pracy, gdzie zaproponowano sprzętowe rozwiązanie, które zapewnia protokół uwierzytelniania oparty o koncepcję wyzwania/odpowiedzi. Przedstawiona metoda wymaga niewielkiej ilości miejsca na układzie scalonym, działa przy bardzo wysokich częstotliwościach oraz jest w pełni kompatybilna z narzędziami projektowania układów scalonych. Chociaż jest on głównie skierowany do układów scalonych wykorzystujących pakietowe przesyłanie danych testowych, proponowana metoda może również poprawić bezpieczeństwo innych złączy testujących.

Metody kompakcji odpowiedzi testowych przedstawione w pracy zweryfikowano eksperymentalnie za pomocą opracowanego przez autora oryginalnego oprogramowania, będącego rozszerzeniem istniejących narzędzi komercyjnych. W eksperymentach wykorzystano produkowane współcześnie cyfrowe układy scalone. Nowe rozwiązania kryptograficzne zostały zweryfikowane za pomocą testów statystycznych, w tym opracowanych przez amerykański National Institute of Standards and Technology (NIST) oraz BSI - niemiecką instytucję certyfikującą bezpieczeństwo informatyczne.

# CONTENTS

# FIGURES

# TABLES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| ANF | Algebraic normal form |
| ATE | Automatic test equipment |
| ATPG | Automatic test pattern generation |
| CG | Challenge generator |
| CRP | Challenge-response pair |
| DFT | Design for testability |
| DIST | Deterministic in-system test |
| EDT | Embedded deterministic test |
| FF | Flip-flop |
| HRG | Hybrid ring generator |
| IC | Integrated circuit |
| IJTAG | Internal JTAG |
| IP | Intellectual property |
| JTAG | Joint Test Action Group standard |
| LBIST | Logic built-in self-test |
| LFSR | Linear feedback shift register |
| MISR | Multiple input signature register |
| NIST | National Institute of Standards and Technology |
| NLFSR | Nonlinear feedback shift register |
| NSL | Nonlinear sequential logic |
| SDC | Silent data corruption |
| PC | Pattern count |
| PHRG | Programmable hybrid ring generator |
| PRPG | Pseudorandom pattern generator |
| PUF | Physical unclonable function |
| RoT | Root of trust |
| SC | Stream cipher |
| SDSFF | State-dependent scan flip-flops |
| SoC | System-on-a-chip |
| SSH | Streaming scan host |
| SSN | Streaming scan network |
| TC | Test coverage |
| TRNG | True random number generator |
| VLSI | Very-large-scale integration |

## ACKNOWLEDGMENTS

First and foremost, I would like to express my gratitude to my supervisors Prof. Janusz Rajski and Prof. Jerzy Tyszer. Their unwavering guidance and support during the entire duration of my PhD project were invaluable. Thanks to their expertise and passion for research I grew up as a researcher. I am also grateful for the numerous suggestions and discussions we engaged in. The successful completion of this research would not have been possible without their assistance and dedication at every step.

I am also thankful to Dr. Grzegorz Mrugalski of Siemens Digital Industries Software, Poznań, Poland, and Dr. Nilanjan Mukherjee of Siemens Digital Industries Software, Wilsonville, OR, USA. Their insightful advice and experience with industrial tools were instrumental in resolving numerous technical problems. Their mentorship was really influential in enhancing the quality and relevance of my research. I also extend my thanks to Dr. Maciej Trawka and Dr. Yingdi Liu, that have been very supportive throughout my PhD journey. Finally, I would like to acknowledge the scholarships from Siemens and the Doctoral School of Poznań University of Technology. In particular, that founding let me participate in international conferences and played a crucial role in making my PhD studies possible.

# 1. Introduction

As can be easily shown, beyond any doubt, the semiconductor industry is ever evolving, with new integrated circuits (ICs) and their updates coming out all the time. For more than half a century, Gordon Moore's eponymous law [119] has well described (and helped drive) steady and staggeringly fast progress in computing technology. This unprecedented pace of micro-electronic miniaturization has led to billions of tiny transistors put on single silicon dies. As ICs are becoming increasingly complex and densely structured, so are physical imperfections whose likelihood of occurrence within ICs is raising alarmingly. They cause defects of various types that may compromise circuits, have a detrimental impact on design performance, and inevitably result in system malfunctions. In order to deliver reliable products, vendors need to test ICs during their manufacturing. In the vast majority of cases, however, it will not suffice to ensure that microchips will function properly throughout their expected lifespan. Indeed, in a wide range of industries, including automotive, healthcare, telecommunications, space, defense, and consumer electronics, it is mandatory to thoroughly test designs during system operations to avoid errors attributed to, for example, post-deployment silicon aging. In another application area, cloud service providers have reported so-called silent data corruption errors caused by subtle IC defects escaping manufacturing tests and producing faulty results only occasionally which makes them extremely difficult to find. It appears that the roots of these sporadic software failures have been traced to timing-related faults in hardware where the performance of transistors may change with varying environmental conditions while running application software. It has raised a call for not only high quality manufacturing tests but also in-system and in-field tests of comparable quality.

Traditionally, deterministic structural tests are used to achieve high quality in chip manufacturing, whereas in-system tests rely on built-in self-test (BIST). Unfortunately, the test quality BIST attains may not be sufficient, primarily due to the pseudorandom test patterns it deploys. Although test points and reseeding of test generators may be helpful in certain cases, logic BIST is usually unable to reach fault coverage visibly higher than 90%. Moreover, an acceptable fault coverage by virtue of pseudorandom patterns can only be obtained for certain types of failures. On the other hand, conventional automatic test pattern generation (ATPG) is capable of working with much more comprehensive fault models (see, for example, cell-aware tests), and it typically achieves a near complete fault coverage. Nevertheless, while automatic test equipment (ATE) employed during the IC production phase can easily handle deterministic test data, the cost of storing the same data directly on a chip is often unacceptable. This dichotomy eventually gave rise to a new technology – deterministic in-system test (DIST) – that combines the quality of ATPG-produced stimuli with a BIST-like paradigm of on-chip test application and test response evaluation. It satisfies both in-field and high-quality test requirements, necessary to ensure reliable operations of ICs throughout their lifespan.

DIST begins to play an essential role in safety-critical applications, in large data centers, or in monitoring silicon aging, to name just a few. These ecosystems require periodic, high-quality tests to assure desired test coverage and short test application, especially in designs that must test themselves when operating. For deterministic tests to be in-system applicable, multi-million-bit test responses with often unavoidable unknown (X) values have to be reduced to small signatures. Typically, X states degrade test results, and thus test response compaction schemes must be duly protected against their negative impact. This is especially true for time compactors, such as multiple-input signature registers (MISRs), whose feedback allows X's to quickly proliferate. Since contaminated signatures render test useless, test response compactors require some form of shielding. As a response to these challenges, this thesis presents two X-masking solutions. The first one is designed to work with a novel logic BIST scheme that features a per-cycle capture mode [118]. The second technique has been customized  to handle test responses in DIST-like environments.

Thanks to deterministic stimuli and optimized X-masking circuitry, DIST can ensure very high test quality, also for complex SoC designs. Unfortunately, the very same test solutions may enable malicious activities. Scan-based attacks are considered a serious threat [46] [181], even though test compression and the Streaming Scan Network (SSN) technology [40] can partially combat security concerns by, for example, scrambling test data. Other countermeasures aimed at protecting test interfaces raise concerns regarding their complexity, both in terms of silicon area and the impact on a design flow. To address these concerns, the second part of the thesis presents new security primitives that can be used to create a hardware root of trust (RoT) capable of defending test infrastructures, specifically those based on SSN.

The remainder of the thesis is organized as follows. Chapter 2 provides a brief overview of the state-of-the-art X-masking solutions. Having defined requirements for a reliable test response compaction scheme, Chapter 3 presents maXpress, a new modular X-tolerant compactor that is applicable to LBIST with the observation scan technology [118]. In particular, it employs dedicated selectors and scan gaters to mask unknown states within redefinable groups of scan chains and during designated scan shift cycles. To limit additional test data volume, the scheme allows a predefined number of patterns to share the same control settings. X-masking discussed in Chapter 4, built on certain modules of maXpress, is tailored to a DIST environment. Here, the masking controls are reloaded once per pattern and decoded using an Embedded Deterministic Test (EDT) [146] decompressor. Additionally, the scheme involves the next level of masking that requires a very small amount of variables to observe most of the easy-to-detect faults. Chapters 3 and 4 are complemented by experimental results obtained for both schemes and several large and complex industrial designs.

Chapter 5 opens the second part of the thesis. It brings back common security issues implied by IC testing and recalls techniques developed to secure test infrastructure. It also briefly reviews state-of-the-art cryptographic hash functions and stream ciphers. Chapter 6 is devoted to hybrid ring generators (HRG) – structurally enhanced ring generators [121].

Although HGRs can be used as efficient test response compactors and programable pseudorandom pattern generators, they have been primarily designed as key components of a new lightweight cryptographic hash function introduced in Chapter 7 and a test data stream cipher presented in Chapter 8. Chapter 9 describes a new lightweight true random number generator that leverages the benefits of both the timing jitter of a single multiple-output ring oscillator and a high-speed ring generator (or a hybrid ring generator). New cryptographic primitives of Chapters 7, 8, and 9 are comprehensively evaluated using a variety of statistical tests, including the NIST and AIS-31 test suites. Finally, a hardware root of trust destined for the SSN-based designs is presented in Chapter 10. It takes advantage of security primitives proposed in the previous chapters and provides scalable and secure solutions for the authentication protocol between a chip and a secure server. The thesis concludes with Chapter 11.

# 2. Unknown states and design for testability

The semiconductor industry demand for test data compression has not slowed down since its first introduction to the market in 2001. In fact, test response compaction, in conjunction with stimuli compression, continues to play a crucial role in handling test data volume growth. Although development of compaction schemes reflects ever-changing needs of many application domains, reliable test response compactors are expected to (1) maintain very high compaction ratios, (2) provide ability to detect a variety of failures found in real silicon, and (3) assure design simplicity. This can only be achieved provided a compactor is capable of preserving observability of the vast majority of scan cells for a variety of unknown (X) states, which are increasingly often identified as having potential for rendering test useless. The presence of X states is attributed to uninitialized memories, non-scan flip-flops, bus contentions, floating buses, internal three-state logic, unwrapped analog circuitry, false paths, cross-domain paths, or paths with timing closure problems. X states may also show up due to last-minute timing violations associated with missing constraints, design issues, or engineering change orders. In many scan-based designs, X states, once captured in scan cells, are subsequently injected into a test response compactor where they can severely affect test results. For example, Xs can result in a loss of test coverage (TC) if not handled properly and tend to increase pattern counts required to test a device thoroughly. As it is vital to control compactor operations with a minimal amount of additional data having no negative impact on the effective test compression, this chapter will briefly revisit certain test schemes and methods to

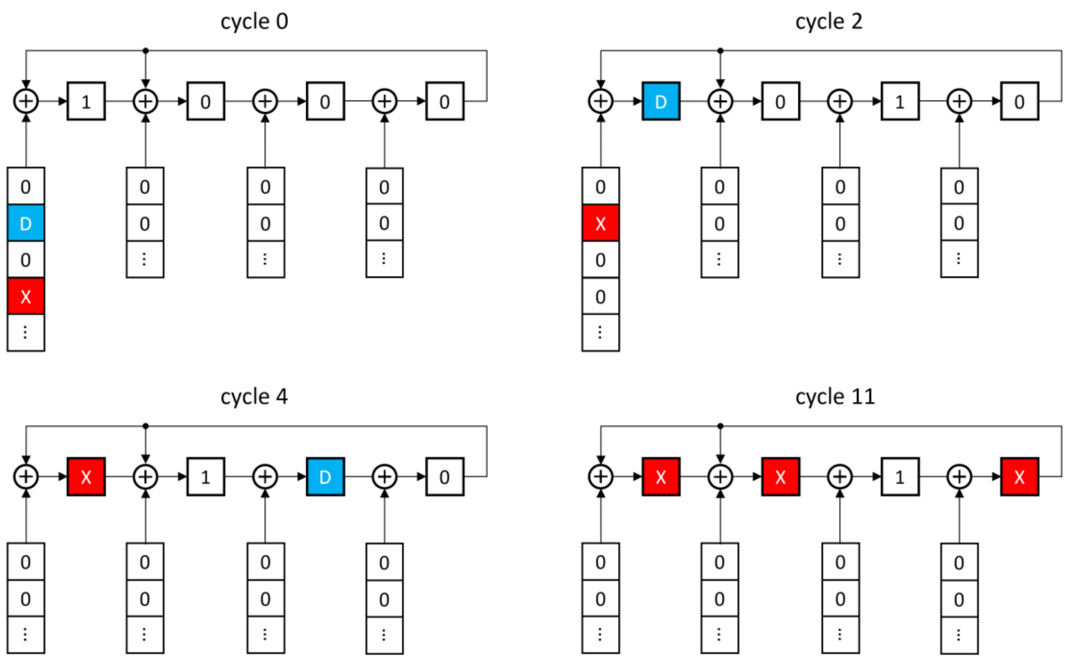


Figure 2.1 X-value circulation in a four-bit MISR.

identify potential areas of improvement for existing schemes that face the future requirements of deep submicron IC testing.

## 2.1 Impact on test response compaction

As already mentioned, unknown states, once injected into test response compactors, may render the outcomes of a test unusable, especially if one deploys a time compactor where X states quickly multiply (due to a feedback fan-out), contaminate a signature, and stay there until a read out operation. Usually, the time compactors are based on linear feedback shift registers (LFSRs) that receive test responses through parallel inputs to finally form a structure known as a *multiple-input signature register* (MISR). Fig. 2.1 is an example of how a single X-value can damage a test response produced by a 4-bit MISR.

First, the faulty effect (D-value[1]) is injected during the second clock cycle. Once an X state enters the MISR in the fourth cycle, it overwrites the faulty effect only after the next seven cycles. Clearly, in this case, the test will not expose any faults. In another outcome, a MISR may contain both X-values and errors. However, every X-state occurring in a final signature doubles the required number of golden signatures (signatures corresponding to a fault-free design to be compared with the signature generated by the compactor). As shown

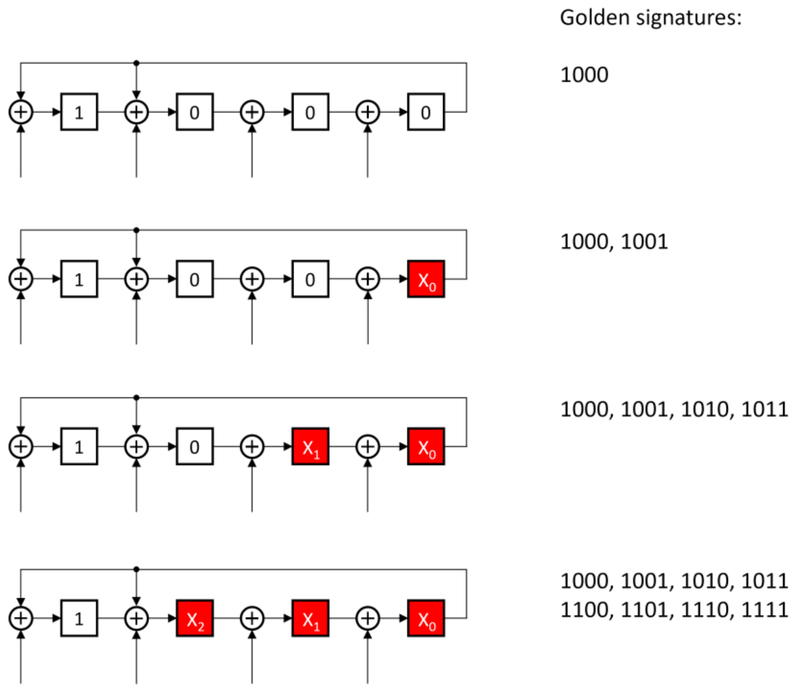


Figure 2.2 Golden signatures for a MISR with 0, 1, 2, and 3 X-states.

[1] The use of term D-value (or D, for brevity) follows the convention originally introduced in a seminal paper of P. Roth on D algorithm [157].

in Fig. 2.2, three X states placed in three different MISR locations would increase the number of potential golden signatures from one to eight. It is therefore essential to ensure that data injected into the MISR or any other time compactor are X free. While in certain cases designs can be X-clean, it is usually necessary to eliminate all X values by deploying an additional X-masking logic.

In contrast to time compaction, its combinational counterparts do not employ memory elements to collect test responses but process them by means of, for example, XOR trees [28]. An example of an eight-input and three-output combinational compactor is shown in Fig. 2.3a. Three XOR gates are used there to compact responses from eight scan chains to obtain a 2.67x compaction ratio, i.e., the ratio of the number of scan chains and the number of compactor's outputs. Even though responses are not accumulated timewise, Xs can still dominate errors when observed in the same cycle. Therefore, to preserve D-values, each scan chain must be connected to at least two outputs of a compactor.

Such a solution has been used in the X-compact [114], where results from each scan chain reach three outputs in parallel. As shown in Fig. 2.3b, even though a compactor receives both a single X and a single D, the error can still be observed on two out of five compactor outputs. The X-compact tolerance of unknown values highly depends on the compaction ratio [114]. To maintain its acceptable degree, this scheme can typically handle just a single X within a single cycle. Such tolerance is usually insufficient; hence, there have been many

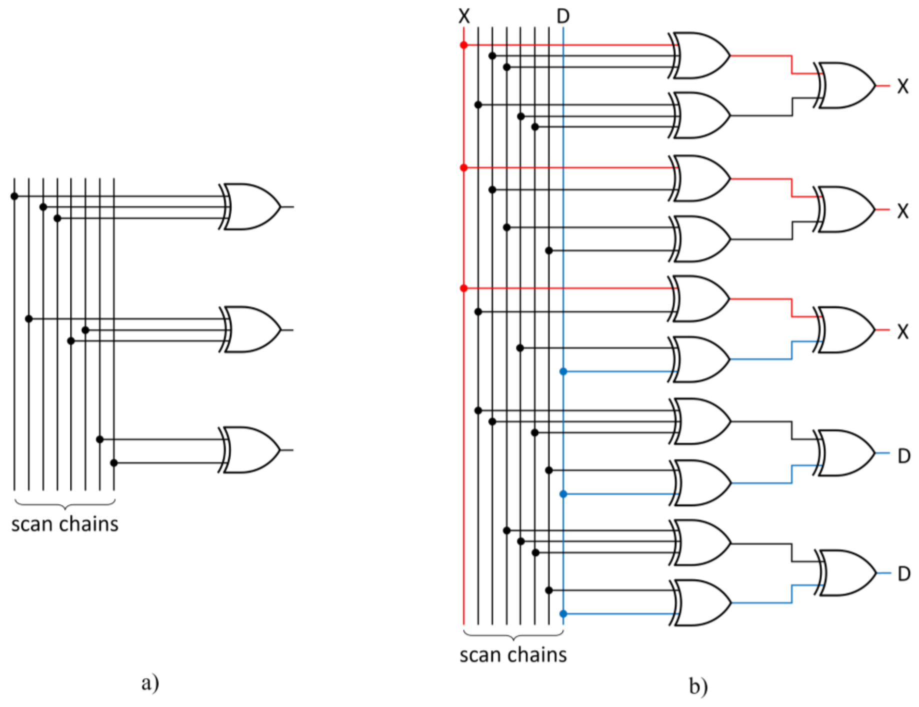


Figure 2.3 (a) Space compaction using linear compactor with eight inputs and three outputs [28]; (b) X-compact [114] with eight inputs and five outputs.

solutions proposed over the years aimed at reducing the negative impact of unknown values on test outcomes.

## 2.2 X-tolerant compactors

As documented by the scholarly literature, several works have been tackling design of so-called X-tolerant compactors. In principle, these devices do not eliminate Xs in their entirety. Instead, they are capable of preserving erroneous results provided the amount of Xs reaching a compactor does not exceed a prespecified upper limit. For example, a convolutional compactor [143], [151] employs a finite memory to buffer data from the scan chains for a few shift cycles. The corresponding design principles, based on the Steiner systems [38], significantly reduce the probability of error cancellation. As shown in Fig 2.4, every scan chain is connected to three different memory elements. Consequently, an error cannot be masked by a single X, and any pair or odd number of errors cannot mask each other. Depending on the convolutional compactor's configuration (and its hardware footprint), more than one X can be tolerated at a time, while the compaction ratios remain relatively high.

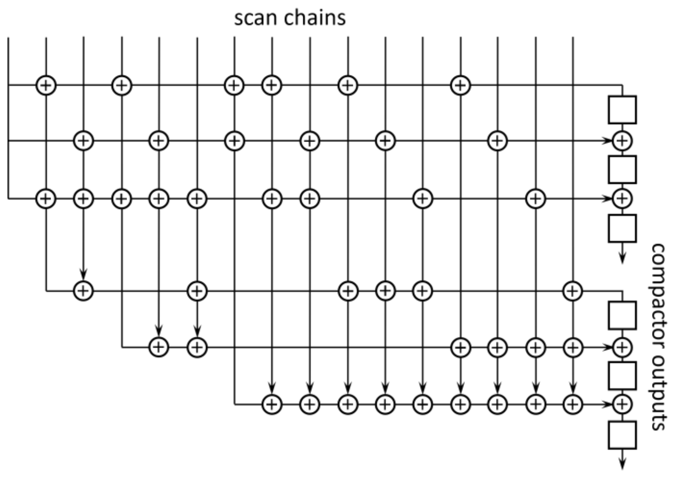


Figure 2.4 Convolutional compactor [151] with sixteen inputs and two outputs.

Another solution [115] combines a weighted pseudorandom pattern generator (PRPG) with an X-tolerant MISR. Here, the signature analyzers are designed with the help of stochastic coding. This method results in high probability of X-masking and is the X-tolerance basis. Yet another MISR-based scheme uses a programmable XOR network [178]. The controls provided to this circuitry are obtained by solving linear equations for a set of MISR bits that, when XORed together, produce an X-cancelling combination. An extra phase shifter placed between scan chains and the MISR reduces a shift correlation of test response values and decreases the possibility of blocking faulty effects as a side effect of X-cancelling. Another group of solutions [130], [166] is based on error-correcting codes, such as Hamming and BCH codes, where probability of error detection in the presence of several Xs could be increased by adjusting a check word width.

While X-tolerant compactors can handle up to several unknown values in a single cycle, they are not designed to withstand a sudden burst of Xs. To resolve this problem and reduce the number of Xs getting into scan chains, one can identify potential X-sources and block them by utilizing X-bounding logic (Fig 2.5). With an additional AND gate, non-scan flip-flops, floating ports, or analog block outputs can be isolated during a test mode. However, not all X-sources can be treated this way. Furthermore, additional gating logic can negatively impact both timing closure and silicon area. Consequently, the presence of X-states in test responses is inevitable.

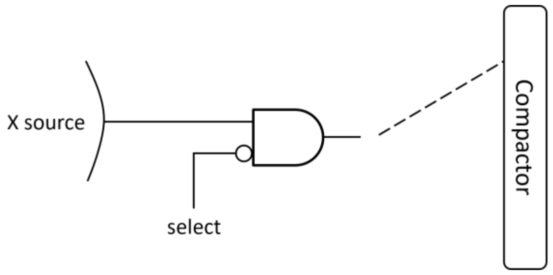
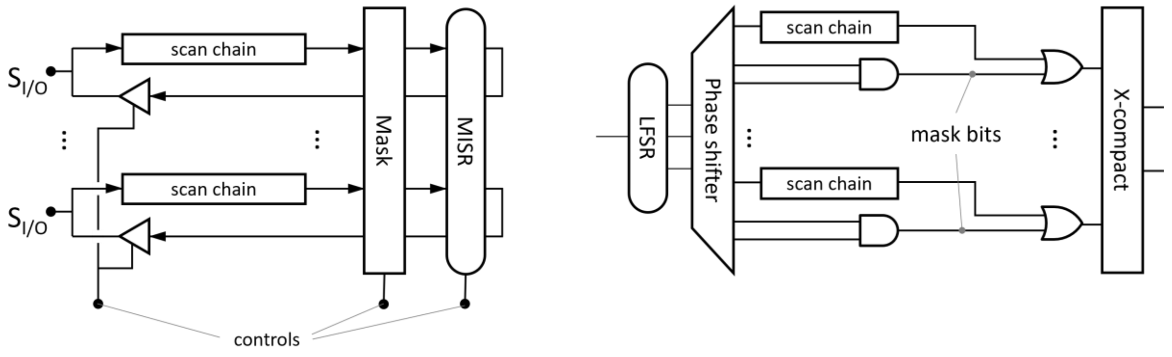


Figure 2.5 X-bounding logic.

## 2.3 Selective masking of scan chains

As demonstrated in Chapter 2.1, time compactors require a protective mechanism to completely block (mask) X states before they reach the compactors' memory elements. Typically, this is accomplished by virtue of schemes monitoring scan chains selectively. Usually, they employ a dedicated circuitry to mask selected unload values so that Xs do not reach a compactor. As a result, the X-masking schemes capable of observing scan chains in a per pattern or/and per cycle manner have been extensively researched for years. The proposed solutions offer tradeoffs between a silicon area overhead, a potential fault coverage drop caused by inadvertent masking of faulty effects, and the amount of additional test data used to control X-masking circuitry. In OPMISR [10], [11], selected unload values can be masked, preventing X states from reaching a MISR. As shown in Fig. 2.6a, external signals are used to control mask logic, a MISR state, and the direction of $S_{I/O}$ pins. The EDT technology [146] uses a selective compactor [172] to mask a given number of scan chains by deploying a register file encoding targeted scan chains. In addition to the masking logic, it uses an enhanced ATPG algorithm capable of handling Xs. In [128], scan chains are gated in a per-cycle fashion thanks to the LFSR reseeding. To minimize the linear dependencies between the masking signals, and to reduce the probability of blocking X-free responses, mask data can be processed through phase shifters and AND gates, as presented in [184]. Fig. 2.6b shows that the



a) b)

Figure 2.6 (a) OPMISR test architecture with bidirectional scan pins [11];
(b) LFSR-based test compactor with mask bits generated through phase shifter and AND gates [184].

mask bits must be set to 1 to replace Xs with known values. Furthermore, the X-compact is used to mask X's that could not be blocked during the previous stage.

The X-Block of [189] uses an LFSR to generate controls for the masking logic. Similarly to the solution of [128], this approach also employs LFSR reseeding to compress masking data. Furthermore, it targets single-detected faults, thus reducing the amount of control data. A hybrid selector presented in [194] combines a PRPG-based method to block Xs every shift cycle with a so-called "Xchains Register" to mask scan chains affected by Xs for several test patterns. It also works with a masking-aware test generation algorithm to target faults that can be observed outside of blocked groups of scan chains. An X-masking logic of [173] is aimed at preserving the coverage of unmodeled defects, and its controls can be provided by any LBIST or test compression scheme. Another test-dependent masking circuitry is proposed in [137], where unknown values are replaced with a known constant by dedicated comparison blocks. A channel masking scheme shown in [35] offers three different channel masking states that either disable all scan chains or select those belonging to one of two groups at the price of possible over-masking. The X-Press scheme (used by EDT) [149],

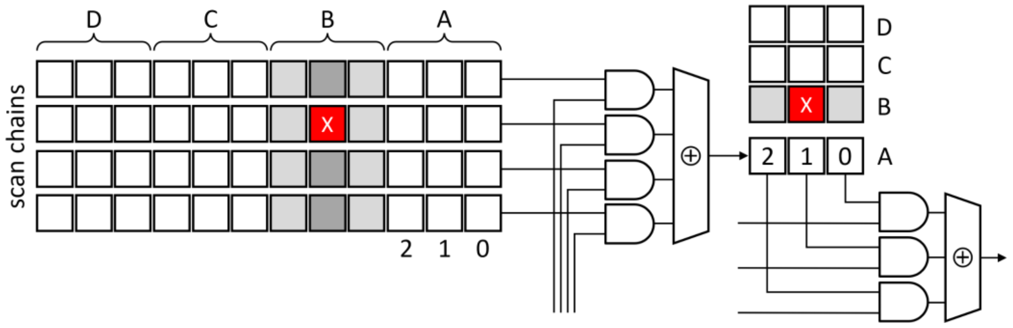


Figure 2.7 X-Press [149] test response compactor.

[150] combines two levels of masking. As shown in Fig 2.7, an X can be blocked in a twofold manner: either by a per-chain masking circuitry or with the overdrive register. Such an approach leads to a high probability of linear independence between mask bits – if X cannot be blocked by the first stage, it might be canceled within the second stage. The controls for both modules are obtained with a scan chain ranking algorithm based on the locations of observation points and X values.

A work presented in [42] can block all X states with a per-cycle resolution and is capable of reusing control data for various test responses. In general, it is based on a finding that many test responses in scan-based designs feature identical or similar X profiles, with Xs grouped in adjacent areas of scan chains. A comprehensive scheme working with PRPG, MISR, and X-masking logic that can be deployed in both scan-based test compression and LBIST (including hybrid solutions) has been presented in a sequence of works [192]–[197]. Recently, a hybrid space compactor has been introduced in [105] that combines a pseudorandom control of a stochastic test response compaction of [115] with a deterministic compaction phase to cope with high X fill rates varying with frequencies of faster-than-at-speed tests [77] used to detect small delay faults. Other techniques to block X states are disclosed in patents; examples may include [25], [126], and [148].

Finally, X-masking schemes for hybrid applications of test compression and LBIST must respond to yet another number of challenges and needs of in-field and in-system test. A major source of complexity in this scenario comes from the requirement to control scan selection with a low amount of data while handling a wide range of static and dynamic X-state profiles. Many unknown states, even if clustered, are typically restrained from capturing by, for example, DFT logic inserted during design implementation. However, the last-minute timing violations can show up anywhere in a design, and the resultant Xs are difficult to block at this very late stage of a development cycle.

## 2.4 Deterministic in-system test and X-masking

Another challenge for X-masking schemes is related to the foreseen importance of the DIST, where deterministic test patterns are combined with in-system test compaction. With LBIST's working only with basic fault models and reaching only around 90% fault coverage, DIST is expected to gain adoption over the coming years. For example, safety-critical devices, compliant with regulations such as the functional safety standard ISO 26262 must thoroughly test themselves during system operations, and should a defect occur, they must put the entire system in a safe state to avoid a system failure. Depending on the safety goal, desired fault coverage may reach over 99%, which may not be achievable by a regular LBIST.

It appears that deterministic-quality test patterns are also needed to handle SDC failures that have recently attracted a lot of interest [47], [54], [80], [165]. Their main symptoms are typically subtle, erroneous computations. When such a failure is not detected, it can quickly spread across several services, leaving no trace or information in system logs about the defect's origin. SDC failures are usually caused by small-delay faults that can be only targeted by timing-aware patterns. Those test patterns must be applied periodically at various stages of a device's life span. That creates new test problems, especially when considering defects manifesting in corner cases or after post-deployment aging. Clearly, a deterministic test is the only known method to guarantee the detection of these types of defects. To be used in-system, the deterministic test should also enable schemes based on the input-streaming-only approach that reduces the volume of test data by employing advanced test compression techniques [83], [96], and replacing all explicit test responses with a MISR-produced signature.

The reduction of test data attributed to a MISR-based compaction is meaningful even within a single circuit. The benefits are even more appreciable when a state-of-the-art X-masking scheme is applied in System-on-Chip (SoC) designs comprising hundreds of cores (often forming 3D stacks). In addition, SoCs may save a substantial number of resources by giving up on output channels that can be reused to strengthen the input test interface. It is a vital commodity when testing SoC designs through, for example, an SSN [40], i.e., a bus-based scan data distribution architecture that enables high-speed test data delivery and facilitates testing of many cores with a constant cost (see also Fig. 2.8, where tests are delivered as packetized scan data on the SSN bus, and streamed through the Streaming Scan Host nodes).

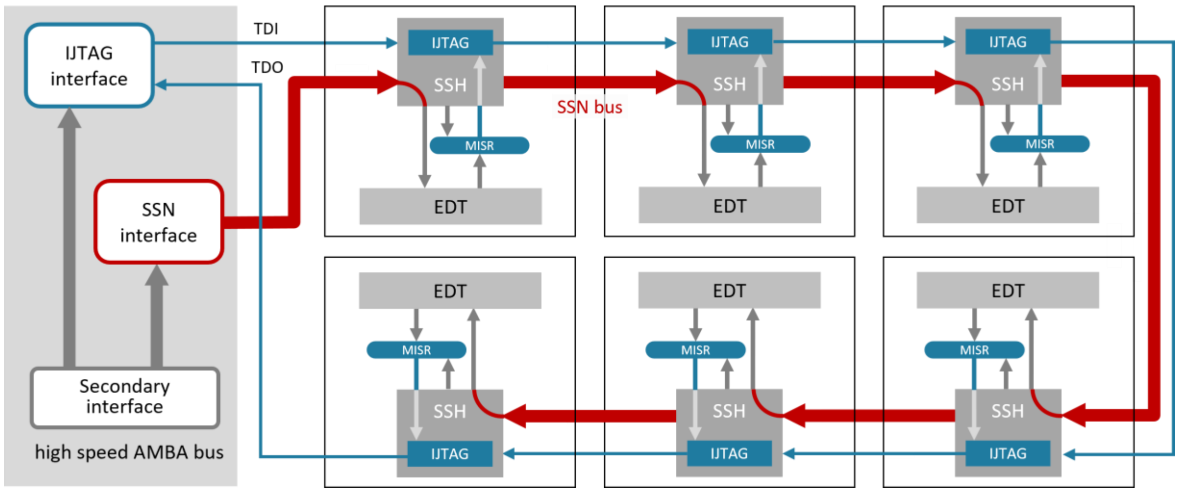


Figure 2.8 Deterministic in-system test setup applied to a 6-core SoC design using SSN [40].

## 2.5 Novel X-masking solutions

As shown in the previous chapters, X-masking hardware is an essential part of many scan selection schemes. It is, therefore, mandatory to reduce its area, especially in contemporary designs with hundreds or thousands of scan chains that require programmable and very flexible selection algorithms. To respond to these concerns, the next chapter presents a new X-tolerant tunable compactor termed *maXpress*. Its *modular* scan selection logic allows masking X states within controllable groups of scan chains and scan shift cycles. The proposed technique helps to tolerate X's discovered very late during the design cycle so that LBIST can still provide a desired TC. Furthermore, to facilitate an in-system test whose TC is achievable in a much shorter time than that of a conventional LBIST, the presented scheme can work with hybrid test points that capture fault effects every shift cycle into flip-flops forming separate observation scan chains [118]. Finally, Chapter 3 presents methods to find the best control parameter settings for the proposed scan selection architecture.

A solution presented in Chapter 3 is intended for low-cost logic BIST applications that typically achieve 90% coverage of simple fault models with a minimal amount of test data to be stored. However, efficient as it is in LBIST, maXpress requires additional and new features to work with DIST. This is because of a gap between LBIST goals and DIST requirements manifested in:

- the ability to target advanced fault models and attain a maximal possible TC (above 99% of testable faults),
- the aggressive test time window paired with the ability to run periodic and frequent tests in real time (especially in the automotive area),
- reasonable test storage – it might be of concern, if there is a need to accommodate not only stuck-at faults, but also transition, delay, or cell-aware failures,
- an option to update tests based on defects seen in field returns or in new technology nodes,
- the ability to test selected (idle) cores while others are in a functional mode or to perform fault diagnosis with core-level resolution (it applies to multiple-core designs using IJTAG and SSN protocols – see Fig. 2.8).

The X-masking scheme capable of working within the framework of DIST is presented in Chapter 4. While building on a highly scalable, layout-friendly, and test-set independent scan chain selection approach, the new scheme allows one to selectively block all X states. Working synergistically with methods to automate settings of the scheme controls, it offers complete observability of errors based on EDT-encoded test data.

# 3. X-tolerant compactor for observation scan

This chapter presents maXpress - an X-tolerant, modular, and programmable compactor [101], [102], deploying a new scan chain selection mechanism capable of completely masking X states, as required by many in-system or one-directional streaming test applications. The proposed scheme also supports separate observation scan chains that, in contrast to conventional scan, capture faulty effects every shift cycle while their content is gradually shifted into a compactor that also receives values from regular chains. In addition to a new layout-friendly architecture, algorithms to automate control settings based on scan chain selection rules deployed to suppress X states are presented.

## 3.1 Compactor's circuitry

The proposed scheme is a part of an on-chip test environment with multiple scan chains. It is inserted outside the design core and consists of a few blocks, as shown in Fig. 3.1. A test response compactor consists of an MISR, an XOR tree, and a highly modular X-masking logic (the gray area in the figure) that outputs scan chain gating signals so that X states originating at various scan cells do not reach the MISR while observability of the remaining scan cells is preserved to such an extent that the test quality remains uncompromised.

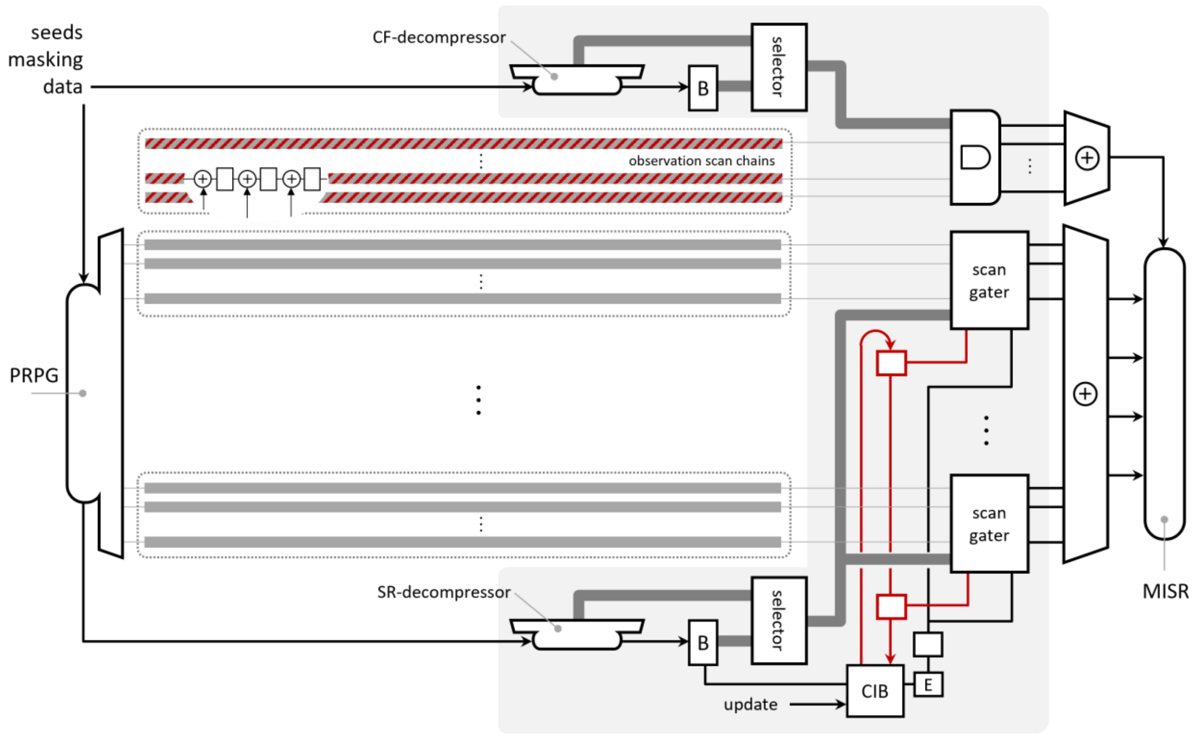


Figure 3.1 maXpress overall architecture.

The actual masking of the regular scan chains is carried out by scan gaters - devices located between the internal scan chain outputs and an MISR, often driven by an XOR tree. Essentially, these devices partition scan chains into disjoint groups of almost equal size, and, if needed, block test results leaving chains within each group before they could enter a test response compaction circuitry.

The scan gater (Fig. 3.2) is comprised of an *n*-bit gating logic, where *n* is the number of scan chains served by a single scan gater. Clearly, if *s* is the total number of scan chains, then the total number of scan groups (scan gaters) is given by $g = s/n$. The actual gating logic is composed of *n* two-input AND gates, and *n* multiplexers. All scan gaters are driven by a common enable line *E* which works with those scan gaters that have not been selected through the configuration register (CR, see below). Setting a configuration flip-flop to 0 disables the corresponding scan gater, and the actual masking value depends exclusively on signal *E*. If $E = 1$, then the scan gater remains transparent and channels test responses directly to the compactor, thus making the associated scan chains fully observable. Having *E* set to 0, however, blocks all scan chains linked with the disabled scan gaters. Two auxiliary flops supply signal *E*, as shown in Fig. 3.1. One of them acts as a shadow register that saves the current value of *E* while reloading its new content in parallel with a shift-in of the next data.

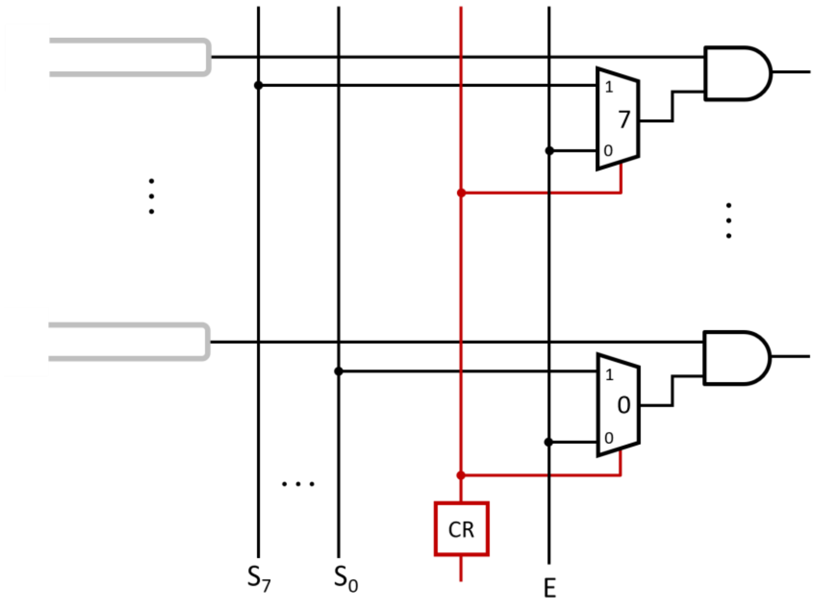


Figure 3.2 Scan gater serving $n = 8$ scan chains.

The scan gaters are driven by a *g*-bit CR, a red-colored item in Fig. 3.1. The purpose of this device is to enable any desired combination of scan gaters. Since every flip-flop of CR serves a single designated scan gater, it eliminates additional address registers and the corresponding converters. Furthermore, there is no need to use multiple-input gates within scan gaters, and there is no need to broadcast scan gaters address data. The CR is daisy-chained with the remaining on-chip test instruments discussed in the following. The CR content is typically shared by several test patterns in a row. As a result, it is reloaded only occasionally and thus is sequestered behind a configuration insertion bit (CIB), as shown in Fig. 3.3 in the spirit of the IEEE 1687 (IJTAG) standard whose segment insertion bits (SIBs) allow access to embedded instruments of reconfigurable scan chains [206]. CIB allows on-demand access to the CR and interfaces the same register with a single channel used to seed PRPG, and to deliver other test-related data. If the CIB flip-flop is set to 0, CIB is set up to bypass the CR and allows only registers B (see Chapter 3.2) and *E* to be updated. Indeed, the CR is unaffected in this mode by any data transfer due to clock gating. Once asserted, the CIB flip-flop routes test data to the CR, at the same time enabling a clock signal to facilitate a shift functionality of the CR. The shift path is established from the input channel, through the register B, into the CR, and then the register *E*. All changes in the CIB flip-flop status are done through the CIB enable input that allows to capture the first control bit of the input sequence. This bit indicates whether the following sequence is to update the X-masking logic configuration, or it is just a new content of other registers.

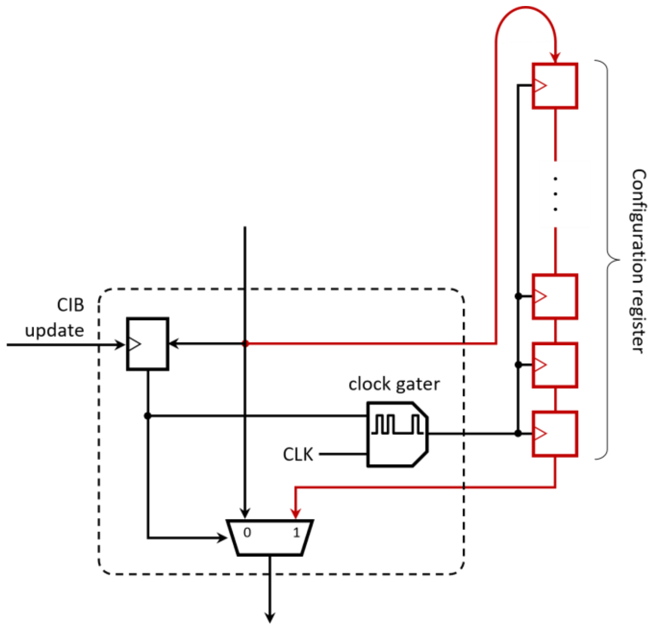


Figure 3.3 Configuration insertion bit.

Enabled scan gaters (their configuration bits are set to 1) receive the actual masking values from a selector through a bus *S*, as shown in Figs. 3.1 and 3.2 (here *S* accommodates 8 bits). As can be seen, each scan chain can be individually blocked provided the corresponding selector output is 0. The selector outputs are shared by all scan gaters, so having several gaters enabled results in blocking the corresponding scan chains in all involved groups. It is worth noting that the scan gaters and the selector lend themselves very well to scenarios demanding very aggressive masking of scan chains for the purpose of observing very few or just a single scan chain. Fig. 3.4 shows a simple summary of the maXpress control settings for a single scan gater. We begin with a CR bit CR. If set to 0, then all scan chains of the associated group are either masked ($E$ = 0) or connected to a compactor ($E$ = 1). When CR = 1, the masking status is decided by the selector such that a given scan chain k is blocked provided $S_k$ = 0, or observed otherwise ($S_k$ = 1).

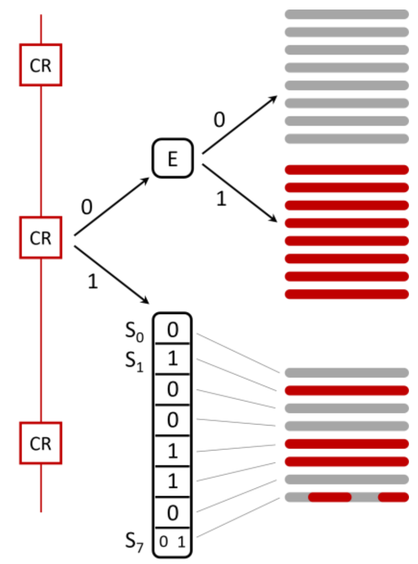


Figure 3.4 maXpress controls for a single scan gater.

To reduce pseudorandom PCs, maXpress may also work with observation scan chains that capture faulty effects, provided by hybrid observation test points, every shift cycle [118], [123]. Essentially, the observation chains accumulate test responses using XOR gates placed in the front of their scan cells (Fig. 3.1) in a manner similar to that of convolutional compactors. It allows one to encapsulate shift and capture functionality within a single clock cycle. The content of observation chains is continuously shifted into the compactor shared with the remaining chains. The observation scan cells do not drive any gates of the original design to prevent sequential dependencies between subsequent patterns occurring in these chains and to avoid over-testing. Selection of test points to elevate TC of pseudorandom patterns follows the procedure presented in [118].

Because of their activity, it is fair to expect that the observation chains may capture a significant fraction of unknown states. Consequently, these scan chains should not be grouped, and have to be masked on an individual basis. A scan gater for $n$ observation chains boils down to a set of $n$ two-input AND gates. It receives the masking signals from a selector similar to that of the regular scan chains. However, per cycle selection data driving the selector are obtained in a different fashion due to much more aggressive X-masking requirements. It is worth noting that the presented approach advocates orthogonal handling of regular and observation chains. It makes it much easier to adapt the proposed solution in designs with no observation scan chains.

Table 3.1 The selector settings.

| B-on | B-off | |
|---|---|---|
| 0 | 0 | scan blocked |
| 1 | x | scan observed |
| 0 | 1 | $S_k$ per cycle |

We will now discuss the design of a selector which contains two *n*-bit registers B-off and B-on and combinational logic to control and mask scan chains within enabled groups. Recall that all scan gaters receive the same controls. The selector of Fig. 3.5 assumes that each group consists of eight scan chains, similarly as in Fig. 3.2. There are two groups of the selector inputs. The inputs denoted as *S* gate individually the corresponding scan chains in a per-cycle mode unless the content of registers B-off and B-on decides otherwise. Indeed, each scan chain is assigned a pair of control bits that determine its masking status, as shown in Table 3.1. In response to suitable 0-signals on bits $b_k$ of both registers, the selector output *k* is set to 0, and thus scan chain *k* is blocked for the period of a complete scan unload. Asserting bit $b_k$ of register B-on makes scan chain *k* fully observable. Finally, setting bit $b_k$ of B-on to 0 and bit $b_k$ of B-off to 1 allows masking of scan chain *k* as required by the current status of input $S_k$ (this is illustrated in Fig. 3.4 by the scan chain controlled by both 0 s and 1 s). Note that B-off and B-on registers are updated via blockage inputs driven by a shadow register B, once per pattern or a group of patterns.

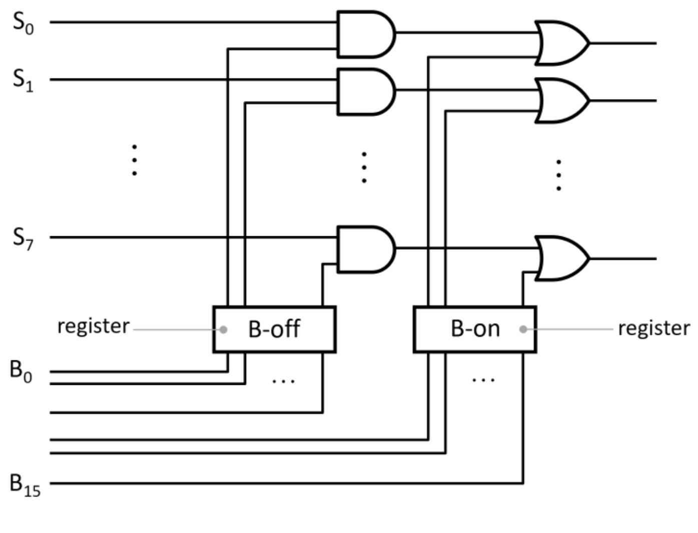


Figure 3.5 Selector block.

It appears that the use of per-cycle controls is either relatively rare or involves a few scan chains only. Consequently, this type of data can be deterministically encoded and provided to the selector by a device referred to as a static-reseeding decompressor, or SR-decompressor for brevity. Its architecture is shown in Fig. 3.6a. Seeds of this small decompressor are delivered through the same single input that is used to initialize PRPG. Besides a ring generator and a phase shifter driving the selector inputs, a hold register is placed between those two devices. It helps in sustaining the selector inputs for more than a single clock cycle, while allowing the generator to change its internal state to ensure encoding of the next group of selection bits. Hence, one can repeat and pass on to the selector a given SR-decompressor state for a number of consecutive scan shift cycles. As is typical of isometric test compression [96], the SR-decompressor houses a circular template register. A 1-signal on its output allows the hold register to have the current content of the ring generator entered as its next state. Because of its size, the very same template is typically used multiple times within the duration of the same test pattern. Note that the ring generator and the template register feature shadow registers that are daisy-chained with the remaining scheme test instruments.

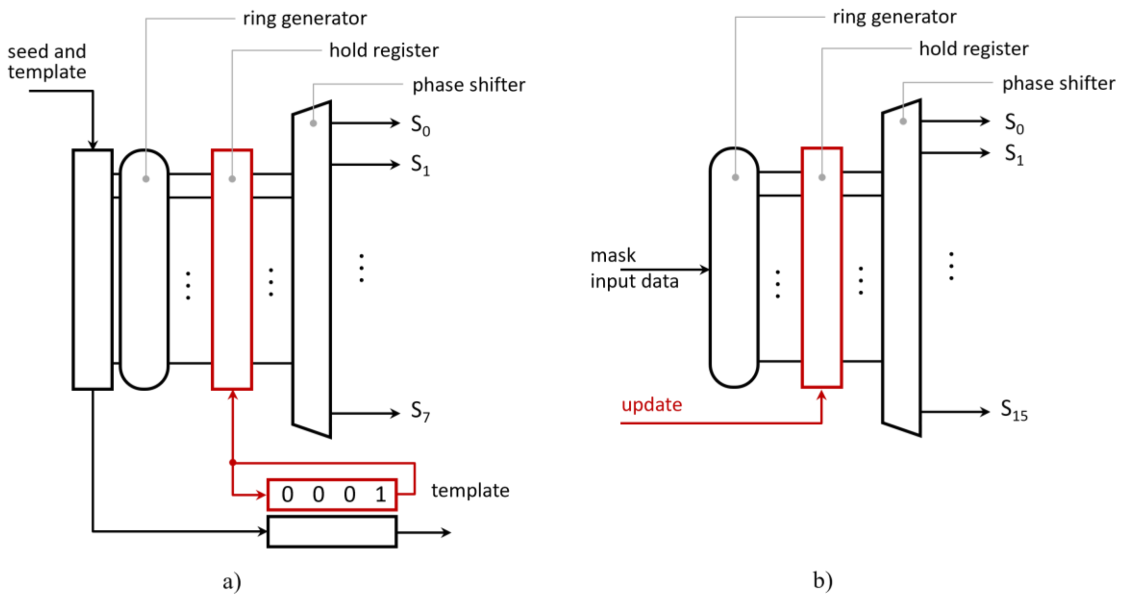


Figure 3.6 (a) SR-decompressor with eight outputs and (b) CF-decompressor.

The SR-decompressor is capable of decoding selection patterns with additional reload points occurring as 1s in the template register to indicate when to update the hold register. It is worth noting that within this framework a selection solver assumes that a single equation is associated with all selection bits corresponding to the same output and covered by a given hold period. As a result, only bits of the period's first cycle become the subject of encoding provided there is at least one such specified selection bit within that period. Experimental results indicate that up to 50% of specified bits (on average) are typically handled by the constant values of the hold periods rather than direct encoding, thus ensuring high encoding efficiency.

A selector working with the observation scan chains has the same structure as that of the selector shown in Fig. 3.5. It uses *m*-bit registers B-off and B-on, where *m* is the number of the observation chains. Similarly, the per-cycle selection data available on its inputs $S_k$ are deterministically encoded and provided to the selector by a test data decompressor. However, in contrast to the regular chains, a continuous flow compression (and hence a CF-decompressor) is used to facilitate encoding of a much larger number of masking signals. Compressed data for the CF-decompressor (stored on chip) are delivered through a separate input (or inputs) along with data feeding the update input of the hold register, as shown in Fig. 3.6b.

## 3.2 Selection of controls

The maXpress performance depends on a method employed to select scan gaters through the CR, and then individual scan chains by means of the selector and signal *E*. Having fault propagation sites and cells that capture X states (X-cells) associated with test responses, the following part now demonstrates how to automate selection of the appropriate maXpress controls. As the presented scheme is to work with LBIST, it has to mask all X's while preserving as many faulty sites as possible.

To reduce test data volume, several successive test responses are expected to share the same control settings. Its selection begins, therefore, by superposing a predefined number of test responses, further referred to as a segment. The output patterns of Fig. 3.7 illustrate this process by using three groups of 8-bit long scan chains, each group comprising four chains. Red-filled circles indicate X-cells, whereas scan cells to which fault *f* propagates are labeled with number *f*. When superposing test responses, the following rules are observed. First of all, the status of X-cells is superimposed on the same cells in the remaining test responses of the same segment even if these cells are fault propagation sites. This is a collateral damage case, and it applies to faults 1, 2, and 3 in responses 1 and 2 of Fig. 3.7 (indicated by three

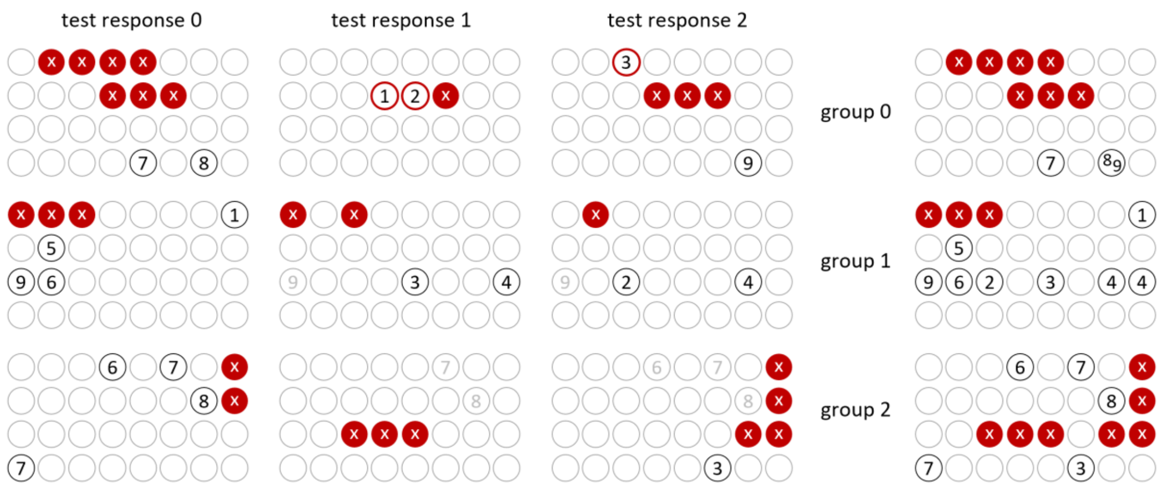


Figure 3.7 Superposition of test responses.

red-circled cells). A fault that propagates to exactly the same cell a few times (for a given segment) is counted only once, as shown by grayed out scan cells that capture faults 6, 7, 8, and 9. However, if a fault propagates to different locations, all such scan cells yield a propagation site count *c* of that fault. The resultant superposition of three test responses of Fig. 3.7 is shown on the right-hand side of the same figure.

The next step sets the CR. In terms of TC, this process is dependent on the value of signal *E*. Recall that setting *E* to 1 enables all scan chains that feed the compactor through disabled scan gaters (Fig. 3.4). Consequently, all scan chain groups that host X-cells must be enabled by having their CR flip-flops set to 1. It allows the actual X-masking by means of the selector module. CR flip-flops of the remaining groups are reset, as shown in Fig. 3.8 where, for the sake of more comprehensive illustration, there are two extra X-free scan chain groups appended to the result of Fig. 3.7. Since the same selector-produced data drive all enabled scan gaters, the corresponding groups are merged once more. The X-cells are again superimposed on the corresponding cells hosted by other scan chains.

The first row of the masking pattern (the right-hand side of Fig. 3.8) contains six X-cells and a single site that captures fault 7. As fault 7 is also observed by two other sites in the last row, the first row can be blocked by setting bits $b_0$ of the B-off and B-on registers to 0, thus effectively gating the first chains of groups 0, 1, and 2. Furthermore, as there are no X-cells in the last row, setting $b_3$ of B-on to 1 lets the fourth chains in groups 0, 1, and 2 to

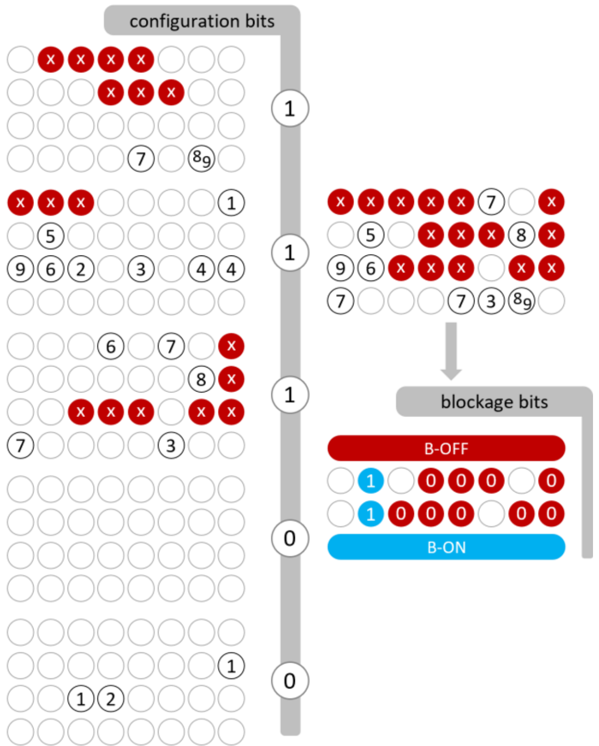


Figure 3.8 Finding maXpress controls for *E* = 1.

be observed in their entirety. Meanwhile, the two internal rows feature a mixture of X-cells and fault propagation sites. These rows are handled by the selector (recall that bits $b_1$ and $b_2$ of the B-off and B-on registers must be set to 1 and 0, respectively) in such a way that X-cells are masked, whereas faults 5 and 6 are let to go. There is no need to encode separately selection bits for faults 8 and 9, as these faults are observed in the last row. Savings like this one increase the likelihood of successful encoding of other selection bits.

As can be easily verified, the above assignments let all faults but four enter the compactor. However, we also need to consider another scenario where $E$ is set to 0. Within this control, all scan chains governed by disabled scan gaters are blocked. Scan chains with X-cells can be, therefore, masked either by means of the selector or by disabling certain scan gaters, while having $E$ set to 0. This dual functionality may allow us to observe more failing scan cells than those of the approach presented previously (with E set to 1). For that, however, one would need to examine $2^g$ different setups of the CR, where $g$ is the number of scan gaters. And only after that one can decide which selection of enabled/disabled scan gaters delivers the best performance of maXpress. For the running example, consider the CR shown in Fig. 3.9 that disables scan gaters 0 and 2. Hence, all scan chains of groups 0 and 2 are blocked because of $E$ = 0. The remaining groups are processed by applying the previously mentioned rules. As a result, faults 1–6 and 9 are observed. Note that in this case, there is no need to employ individual selection bits at all.

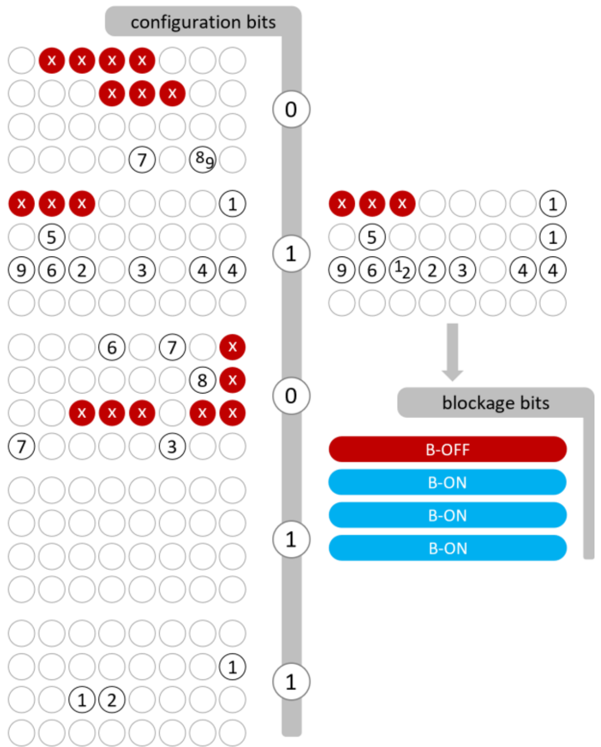


Figure 3.9 Finding maXpress controls for $E$ = 0.

## 3.3 Weights assignment

Given hundreds of thousands of scan cells and millions of faults, a slow processing of long fault lists associated with scan cells (or scan cell lists associated with faults) is needed to determine the actual TC for a given content of the CR. To alleviate this problem and to avoid an exponential growth of the CR states, this register is set up through a greedy approach resembling a hill climbing paradigm. Moreover, lists of faults for propagation sites are replaced with the corresponding fault weights, and then weights of scan cells.

A given fault's weight *w* is the inverse of its propagation site count *c*, that is $w = 1/c$. This weight is linked with all different propagation sites of the fault. Within the superposition of test patterns, a scan cell receives a weight obtained by summing up all individual weights of faults propagating to this particular site. For example, fault 4 (Fig. 3.7) propagates to two different scan cells, so its weight equals 0.5. Similarly, fault 7, as reaching three different scan cells, gets 0.33. The result of superposing of the three responses of Fig. 3.7 in terms of weights is shown in Fig. 3.10. The last but one cell in the fourth scan chain gets the value of 1 = 0.5 + 0.5 as a sum of weights associated with faults 8 and 9. X-cells get the 0-weight.

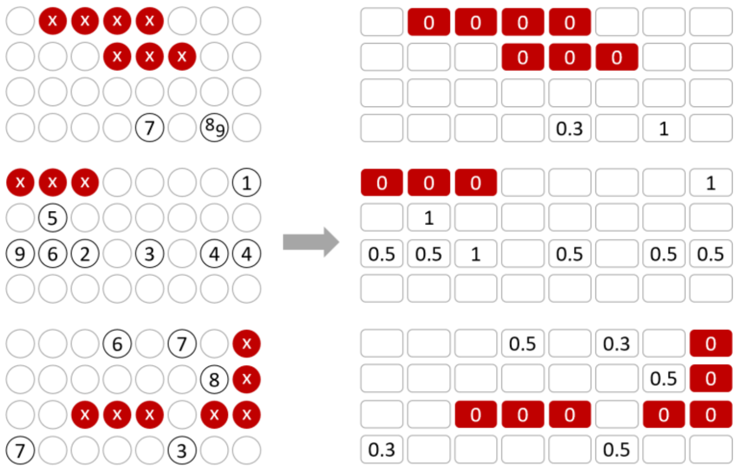


Figure 3.10 Scan cell weights.

Groups of weighted scan cells can be superposed in such a way that X-cells force 0-weights on the corresponding cells in other groups, whereas weights of the remaining sites are obtained by summing up weights of the corresponding cells. Finally, the resultant group weight is equal to the sum of individual weights over all cells of that group. For example, the weights of the groups of Fig. 3.10 are equal to 1.3, 5.5, and 2.1, while the weight of a superposed group is going to be (as can be easily verified) 4.9. This metric is used as a primary figure of merit to assess the performance of the scheme in the selection settings procedure presented in the following passage.

As soon as all cell weights are determined, finding the best setup of the CR proceeds as follows. Let $\mathbf{H} = [h_{g-1} \ldots h_1\ h_0]$ be a *g*-bit binary vector representing the CR, i.e., $h_k = 1$, if *k*th stage of the register is set to 1. Starting with its randomly selected initial state, the first step is to compute a weight of the resultant group obtained by superposing groups associated with the enabled scan gaters, i.e., those for which $h_k = 1$. Subsequently, the following process iterates. The current state of **H** is inverted, one bit at a time, to find the resultant weight of groups being superposed. Note that throughout this step, either an additional scan gater becomes enabled or a so-far-active scan gater gets disabled. Having determined the weights for all $g$ inversions, we pick, in a greedy fashion, vector **H** with a bit whose flipping yields a weight higher than do any of the other bits. Then the procedure is repeated for the current state of **H** except the bit whose inversion was the most effective in the previous step. Although one may need to compute weights up to $g^2$ times, computations for all inversions are independent, and therefore can be easily run in parallel. The selection process continues until the method fails to produce a weight higher than the best value obtained so far and representing the current state of **H**. Typically, the above selection process is repeated a number of times, as a final result may depend on the initial (random) state of **H**. Given the ultimate solution, the obtained TC is used to compare this scenario against the approach with $E = 1$.

In summary, it is immediately clear that every segment of test responses is deployed to arrive with a separate set of maXpress control settings. Fig. 3.11 sketches out the maXpress controls selection flow. First, all response patterns a segment is comprised of are superposed. For $E = 1$, it suffices to determine all groups that feature X-cells, set the CR flip-flops of these groups to 1, disable the remaining scan gaters, and merge all enabled groups into a single one. That allows computation of TC, i.e., the number of faults whose propagation sites have not been canceled out by X values. For $E = 0$, the greedy optimization procedure is used to determine the best setup of the CR with respect to TC. Having found the coverage numbers for both scenarios, we pick the one that yields better result and proceeds with finalizing the selector setup by assigning the appropriate values to the B-off and B-on registers, as well as by choosing the individual selection bits, whenever it is necessary. If the isometric compression is unable to encode all selection bits because of constraints imposed by available seed variables and the size of the test template, one should reduce the number of selection bits by gating the least crucial scan chains through setting the corresponding bits of registers B-off and B-on to 0.

Interestingly, a large body of experimental evidence shows that a rate of setting $E$ to 1 versus having $E$ de-asserted is clearly a circuit-dependent factor, and it is primarily determined by scan architecture, the number of the scan gaters, and the last but not least—the distribution of X states across test responses. Back to Fig. 3.11, if there are still test responses, the method goes back to form a new segment of patterns and to create their superposition. Otherwise, the procedure terminates.

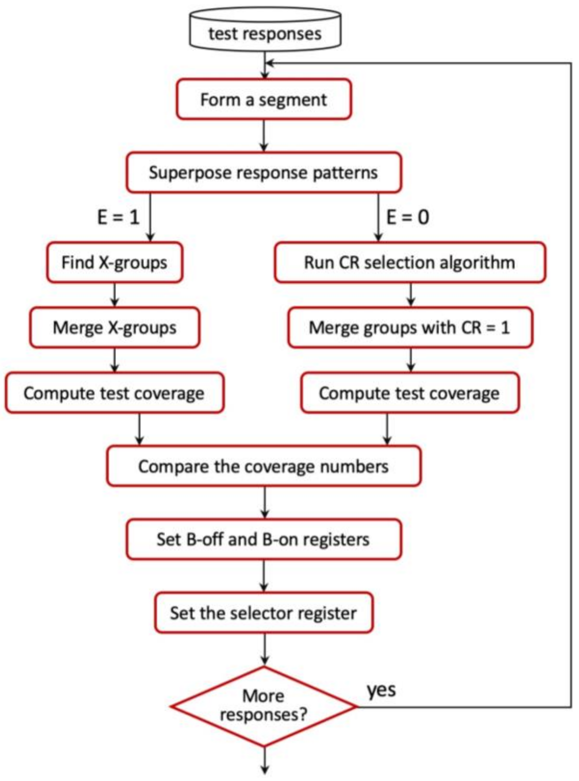
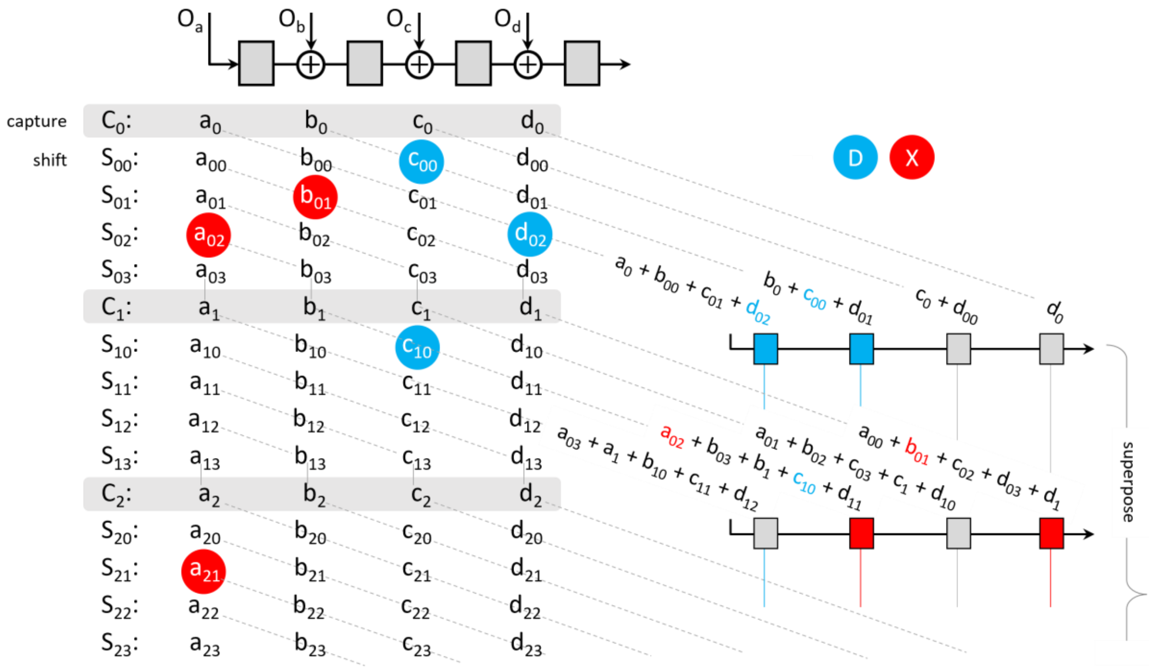


Figure 3.11 Control setting selection flow.

## 3.4 Masking observation scan chains

Whereas regular scan chains may capture X states only once per pattern, the observation scan chains can do that repeatedly in a per-cycle mode. As a result, a single X value may invalidate several faulty effects on its way to the serial output of a scan chain. This is illustrated in Fig. 3.12 for a 4-bit observation chain.

The left-hand part of the figure lists variables representing a circuit response that enters the observation scan either during regular capture cycle $C_k$ (single-index variables $a_k$, $b_k$, ...), where $k$ = 0, 1, 2, ..., or during scan shift cycle $S_{kj}$ (double-index variables $a_{kj}$, $b_{kj}$, ...), where $j$ = 0, 1, 2, 3. The variables are gradually XOR-ed, as indicated by the diagonals, to yield the final responses shown on the right-hand side of Fig. 3.12. Each variable represents either an X state (red circle) or a fault ID (blue circle). It is ignored if nothing has been injected into the observation scan at a particular location and time. Clearly, XOR-ing an X with another combination of variables yields X. Adding several fault IDs produces a list of faults being observed at the corresponding stage of the observation scan unless two fault IDs are identical—they cancel each other as a result of aliasing. Successive responses, corresponding to a given segment of test patterns, are then superposed, as shown in the figure, using the same rules as stated above (except of having the same ID fault several times; such a fault is counted only once as there is no error aliasing in the superposition process).



$O_a$
$O_b$
$O_c$
$O_d$
capture
shift
$C_0$: $a_0$ $b_0$ $c_0$ $d_0$
$S_{00}$: $a_{00}$ $b_{00}$ $c_{00}$ $d_{00}$
$S_{01}$: $a_{01}$ $b_{01}$ $c_{01}$ $d_{01}$
$S_{02}$: $a_{02}$ $b_{02}$ $c_{02}$ $d_{02}$
$S_{03}$: $a_{03}$ $b_{03}$ $c_{03}$ $d_{03}$
$C_1$: $a_1$ $b_1$ $c_1$ $d_1$
$S_{10}$: $a_{10}$ $b_{10}$ $c_{10}$ $d_{10}$
$S_{11}$: $a_{11}$ $b_{11}$ $c_{11}$ $d_{11}$
$S_{12}$: $a_{12}$ $b_{12}$ $c_{12}$ $d_{12}$
$S_{13}$: $a_{13}$ $b_{13}$ $c_{13}$ $d_{13}$
$C_2$: $a_2$ $b_2$ $c_2$ $d_2$
$S_{20}$: $a_{20}$ $b_{20}$ $c_{20}$ $d_{20}$
$S_{21}$: $a_{21}$ $b_{21}$ $c_{21}$ $d_{21}$
$S_{22}$: $a_{22}$ $b_{22}$ $c_{22}$ $d_{22}$
$S_{23}$: $a_{23}$ $b_{23}$ $c_{23}$ $d_{23}$
D
X
$a_0 + b_{00} + c_{01} + d_{02}$
$b_0 + c_{00} + d_{01}$
$c_0 + d_{00}$
$d_0$
$a_{03} + a_1 + b_{10} + c_{11} + d_{12}$
$a_{02} + b_{03} + b_1 + c_{10} + d_{11}$
$a_{01} + b_{02} + c_{03} + c_1 + d_{10}$
$a_{00} + b_{01} + c_{02} + d_{03} + d_1$
superpose

Figure 3.12 Superposition of signals in a 4-bit observation scan chain.

Now the superposed test responses become the subject of the scan chain selection process that partially resembles the approach presented earlier for the regular scan chains. First, we identify X-free observation chains and exclude them from further analysis altogether with faults propagating to these chains. They will be unconditionally observed by asserting the respective bits of the register B-on so as to reduce the encoding burden on the CF-decompressor. For the remaining faults, using their propagation site counts *c*, every cell that captures a given fault is assigned a weight *w* = 1/*c*, as shown in Chapter 3.3 (a fault propagating multiple times to the same cell is counted once within a given segment of patterns). Recall that X-cells receive the value of 0. Moreover, cells that capture neither X nor an error are assigned a small, nonzero weight. The actual encoding phase begins by sorting a list **L** of the nonzero observation scan cells such that they are in order by largest weights. As with the regular scan chains, all per-cycle gating signals are represented by linear functions of mask variables injected into the CF-decompressor. Initially, the set of gating equations is comprised of expressions corresponding to all X-cells. Clearly, the right-hand sides of these equations are assigned the logic value of 0. The main loop of the encoding procedure expands the set of equations by adding an expression corresponding to the first element on the list **L**; then, it tries to solve the current set of equations and removes the first item from the list, continuing until the list is empty. If, at any stage, the equations are not solvable, then the newly added equation is deleted and the corresponding gating signal is discarded. Note that the right-hand sides of all equations taken from the list **L** are always assigned the logic value of 1. It is crucial to observe that equations representing both Xs and errors are not necessarily associated with their actual propagation sites. This is because of how the CF-decompressor hold register is updated.

The encoding procedure partitions a superposed test response into several *blocks* comprising certain number of consecutive clock cycles in such a way that there are no observation scan chains that capture both Xs and errors inside the blocks. It allows one to repeat the same gating signal many times in succession by using the hold register storing a state that the ring generator entered at the beginning of a block. Hence, one can cost-effectively encode identical data such as clustered Xs or multiple errors. The block size is determined by the ability to encode data within its boundaries. The encoding process begins with a block and the corresponding state of a ring generator which should be applied first, and it gradually moves toward the end of a test response. As long as the gating signals can be encoded, the algorithm works by repeatedly increasing the size of the block, and by including successive equations. At some point, a solution may not exist anymore. This particular time frame is then assigned a new block (a reload point), and the procedure continues. It is worth noting that observation scan chains that capture exclusively X states are unconditionally blocked, as shown in Table 3.1. The same rule applies to those observation chains that capture errors, but none of them have been eventually added to the set of gating equations. Finally, observation cells with small weights can also be taken into account, the encoding capacity permitting. It is possible to include unmodeled faults that may propagate to the observation scan chains, as well.

## 3.5 Experimental results

*A. Regular scan chains*

This chapter reports several experiments with eight large industrial designs having all components of the solution on a chip. In this section, the benchmark designs do not use the observation scan technology. The experimental results addressing the presence of the observation chains are presented in Section 3.5B. Table 3.2 lists major characteristics of test cases used in the experiments: the number of gates, the number of scan cells, and the scan architecture. Moreover, Table 3.2 contains the following metrics:

1) the number of control and observation points used in each design, and the number of stuck-at faults;
2) the error-fill rate, i.e., the fraction of scan cells that capture faulty effects, averaged over all test patterns;
3) the total number of scan cells (X-cells) that capture, at least once, X states across all test patterns;
4) the total number of scan chains (X-chains) that capture, at least once, X states across all test patterns;

Table 3.2 Circuit characteristics.

| | Gates | Scan cells | Scan | Test points | | Stuck-at faults | Error-fill rate | X-cells | X-chains | Test coverage | Test coverage w/Xs | Pattern count ISO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | CP | OP | | | | | | | |
| D1 | 1.30M | 76.9K | 1,200 × 65 | 1,029 | 2,362 | 3,925,255 | 4.94% | 82 | 60 | 91.00% | 89.41% | 5,120 |
| D2 | 1.34M | 78.1K | 1,200 × 66 | 976 | 2,479 | 3,876,452 | 5.15% | 83 | 48 | 91.17% | 89.57% | 4,608 |
| D3 | 1.12M | 86.9K | 528 × 169 | 952 | 1,818 | 2,425,160 | 2.34% | 96 | 11 | 91.80% | 89.77% | 4,928 |
| D4 | 1.75M | 119.3K | 729 × 175 | 1,229 | 1,183 | 3,176,430 | 2.54% | 126 | 51 | 91.79% | 87.43% | 3,456 |
| D5 | 2.57M | 194.2K | 817 × 242 | 1,365 | 2,608 | 4,831,590 | 2.14% | 102 | 16 | 92.71% | 91.00% | 2,112 |
| D6 | 6.66M | 294.8K | 1,236 × 242 | 2,329 | 3,603 | 8,967,807 | 2.51% | 105 | 49 | 91.66% | 89.45% | 2,624 |
| D7 | 3.73M | 207.4K | 900 × 237 | 2,771 | 3,731 | 8,480,378 | 5.77% | 107 | 54 | 91.40% | 89.61% | 5,184 |
| D8 | 14.2M | 899K | 3,163 × 291 | 2,418 | 1,600 | 30,513,081 | 5.41% | 317 | 63 | 90.83% | 89.92% | 6,848 |

5) the reference TC of 10 K pseudorandom patterns, recorded at the scan cell outputs, i.e., assuming that there is no test response compaction;
6) the reference TC of 10 K pseudorandom patterns, recorded at the output of a compactor masking individually X-chains for a duration of their entire unload;
7) the pseudorandom PC necessary to comply with the functional safety standard ISO 26262 and its Automotive Safety Integrity Level D (ASIL D) ratings that call for the 90% TC target [79] (again, assuming no test response compaction).

It is worth noting that errors may occur in clusters; local error-fill rates can be therefore higher than the average values. In particular, it applies to the first groups of patterns that have much higher error fill rates than the remaining responses.

Table 3.3 summarizes the results of the experiments with the proposed scan selection logic enabled. For each circuit, the following information is provided:

1) the TC of 10 K pseudorandom test patterns;
2) the PC of tests needed to regain the 90% TC target;
3) the total number of control bits (CB) per segment necessary to operate maXpress; CB multiplied by the number of segments gives the total number of control bits;
4) the second (real) metric $\rho$ in the column CB is the number of segments that maXpress can deploy, if the total number of maXpress-used control bits was the same as that of a reference scheme masking individually $s$ scan chains for a duration of their unload, i.e., $\rho = s/\text{CB}$;
5) average observability (OB+) of scan cells which capture neither X states nor errors; this figure of merit can be used to assess fortuitous TC, i.e., the odds that an unmodeled fault can be detected if it propagates to scan cells which are not observed on purpose.

Table 3.3 Experimental results – the group size $\lceil \sqrt{2s} \rceil$.

| | | The number of segments | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 4 | 8 | 10 | CB |
| D1 | TC % | 70.99 | 75.93 | 84.69 | 90.24 | 90.54 | 162 |
| | PC | 82,048 | 51,264 | 20,544 | 9,024 | 7,232 | $\rho$ = 7.4 |
| | OB+ % | 44.68 | 43.88 | 47.19 | 52.08 | 51.35 | |
| D2 | TC % | 73.76 | 77.66 | 83.45 | 90.51 | 90.77 | 162 |
| | PC | 92,224 | 46,144 | 23,104 | 8,320 | 7,232 | $\rho$ = 7.4 |
| | OB+ % | 51.16 | 51.50 | 46.90 | 57.54 | 60.86 | |
| D3 | TC % | 86.12 | 87.48 | 90.42 | 91.82 | 91.89 | 131 |
| | PC | 30,784 | 20,544 | 8,448 | 5,952 | 5,184 | $\rho$ = 4 |
| | OB+ % | 88.95 | 89.42 | 90.72 | 92.93 | 93.83 | |
| D4 | TC % | 69.51 | 74.46 | 86.90 | 90.58 | 91.00 | 141 |
| | PC | 92,288 | 61,568 | 18,368 | 8,448 | 6,912 | $\rho$ = 5.2 |
| | OB+ % | 62.83 | 62.28 | 67.22 | 65.99 | 72.91 | |
| D5 | TC % | 82.73 | 85.60 | 90.76 | 92.49 | 92.56 | 145 |
| | PC | 71,744 | 30,784 | 7,744 | 3,904 | 3,136 | $\rho$ = 5.6 |
| | OB+ % | 83.99 | 83.46 | 84.63 | 78.97 | 86.18 | |
| D6 | TC % | 64.24 | 71.01 | 83.39 | 90.98 | 91.19 | 165 |
| | PC | > 100K | 66,752 | 20,544 | 7,744 | 6,208 | $\rho$ = 7.5 |
| | OB+ % | 56.38 | 56.76 | 58.22 | 66.93 | 68.70 | |
| D7 | TC % | 72.88 | 78.57 | 85.14 | 90.45 | 90.58 | 149 |
| | PC | 72,128 | 43,200 | 20,544 | 9,024 | 8,256 | $\rho$ = 6 |
| | OB+ % | 62.73 | 61.01 | 55.08 | 69.24 | 66.48 | |
| D8 | TC % | 71.93 | 75.02 | 84.92 | 89.37 | 90.00 | 225 |
| | PC | > 100K | > 100K | 35,200 | 12,224 | 10,240 | $\rho$ = 14.1 |
| | OB+ % | 64.72 | 63.88 | 66.11 | 72.19 | 72.89 | |

In all experiments but one, maXpress deploys a 32-bit SR-decompressor with a 32-bit template. The presented results vary in the number of segments which implies different rates of control reloads. It also determines the amount of test data represented by the number CB of control bits. Recall that CB comprises the content of the following storage elements: the $g$-bit CR, the registers B-off and B-on, the SR-decompressor seed and the test template, plus bit E. Given the number $s$ of scan chains, one needs to store a total of $g + 2s/g + Seed + Template + 1$ bits per load. To minimize the number of flip-flops, one should find a value of $g$ for which the above formula assumes its minimum. As can be easily verified, the desired number of groups is given by $g = \sqrt{2s}$. Consequently, the experiments are run for the number of groups being equal to $\lceil \sqrt{2s} \rceil$. Because of its size, design D8 uses a 64-bit SR-decompressor and doubles the number of groups.

The number of X-cells in designs of Table 3.2 ranges from 82 (D1) up to 126 (D4). Clearly, sites that capture X states as well as the likelihood of this occurring may shape the final results. For example, X states populate quite a few scan chains in almost all designs in a uniform manner across all test patterns, whereas there are only a few X-chains in design D3 where the vast majority of scans has no unknown states at all. It appears that the ability of the scheme to work with different sets of groups (scan gaters) allows maXpress to address these challenges in an efficient way; see results collected in Table 3.3. In particular, with the increasing number of X-chains, the new scheme appears to be much more robust and flexible than conventional solutions (compare the coverage numbers in the last but one column of Table 3.2 and the corresponding results in Table 3.3). Data reported in Table 3.3 indicate that test coverage increases with the increasing number of segments, whereas the corresponding number of test patterns systematically decreases. As a result, one can trade-off these factors against a test data volume (the number CB of bits) necessary to control maXpress with the varying number of segments. It is also worth noting that the over-masking has low impact on the test quality, as shown in rows OB+. Indeed, the average observability of scan cells, including those not directly protected by the X-masking scheme, remains very high and, in particular, guarantees detection of several unmodeled faults.

Fig. 3.13 plots more detailed outcomes for design D4 while observing test coverage and sweeping the number of applied test patterns. The experimental results consist of five curves:

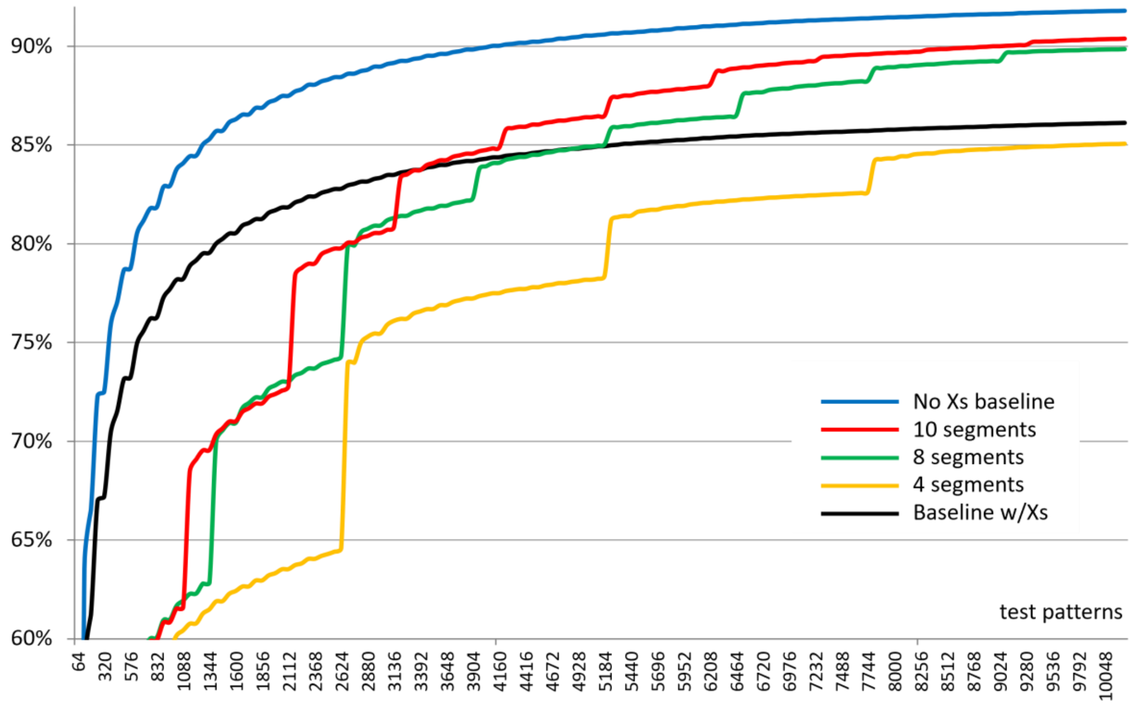


Figure 3.13 Test coverage for design D4.

- the reference test coverage recorded at the outputs of scan cells (a blue curve),
- test coverage observed on the output of a compactor se-cured by maXpress while partitioning test patterns into 4, 8, and 10 segments (a yellow, green, and red curve, respecively) of equal size; recall that the number of segments corresponds directly to the number of maXpress configurations,
- the reference test coverage seen at the output of a com-pactor disabling individually scan chains for the entire period of their unload (a black curve; see also Table 3.2).

Clearly, significant parts of curves representing the reference coverage and that of 10 segments lie close to each other. Given a minor (or in some cases negligible) difference between these two cases, one may conclude that the new scheme offers very good observability of scan errors even in the presence of a gross number of X states. In other words, maXpress does not compromise test quality, and this is accomplished in a very cost-effective manner – see the amount of test data needed to control the scheme for each test case. This trend occurs systematically across all designs, including those not reported here.

For the sake of summary, Table 3.4 brings back the key results presented in this chapter. Besides two reference test coverage metrics of Table 3.2, it lists three maXpress-produced outcomes: (1) test coverage achievable for the same amount of control data (TCCD) as that of the scheme masking individually *s* scan chains for a duration of unload (it needs *s* control bits), (2) the best test coverage (BTC) achieved (typically for 10 segments), (3) a difference $\Delta$ between BTC and the second reference. Clearly, maXpress outperforms the conventional X-masking scheme and comes close to the upper bound results of an approach (column "no compaction") where test coverage is recorded without a compactor.

Table 3.4 Test coverage [%] summary.

| | References | | maXpress | | |
|---|---|---|---|---|---|
| | no compaction | w/Xs | TCCD | BTC | Δ |
| D1 | 91.00 | 89.41 | 90.24 | 90.54 | 1.13 |
| D2 | 91.17 | 89.57 | 90.51 | 90.77 | 1.20 |
| D3 | 91.80 | 89.77 | 90.42 | 91.89 | 2.12 |
| D4 | 91.79 | 87.43 | 86.90 | 91.00 | 3.57 |
| D5 | 92.71 | 91.00 | 90.76 | 92.56 | 1.56 |
| D6 | 91.66 | 89.45 | 90.98 | 91.19 | 1.74 |
| D7 | 91.40 | 89.61 | 85.14 | 90.58 | 0.97 |
| D8 | 90.83 | 89.92 | 90.00 | 90.00 | 0.08 |

## *B. Observation scan chains*

Essentially, experiments described in the following section are parallel to those of the previous section. This time, however, the same designs as before (industrial cores) deploy a certain number of observation scan chains replacing a similar number of the regular chains. As adding the observation scan to an original core changes its test-related functionality, in particular locations and breakdown of test points (note that almost all observation points are now feeding exclusively respective flip-flops of the observation chains), Table 3.5 lists all relevant characteristics again. Note that the reference TC metrics reported in Table 3.5 have the same definition as that of Table 3.2, including observation scan cells. The number of faults is increased due to additional logic associated with every observation scan cell. The basic data of test cores are already available in Table 3.2. Table 3.5 contains, though, additional characteristics *d/v* of the CF-decompressor, where *d* is its size, and *v* gives the number of input channels (it accounts for a separate input feeding the hold register). Furthermore, labels "X-cells" and "X-chains" refer exclusively to the observation scan chains.

Table 3.5 Circuit characteristics with observation scan.

| | Observation scan | CF decompressor | Test points | | Stuck-at faults | Error-fill rate | X-cells | X-chains | Test coverage | Test coverage w/Xs | Pattern count ISO |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | CP | OP | | | | | | | |
| D1 | 34 x 63 | 32 / 3 | 1,278 | 2,113 | 3,943,552 | 54.92% | 1194 | 32 | 93.82% | 89.20% | 512 |
| D2 | 39 x 64 | 32 / 3 | 1,367 | 2,088 | 3,953,440 | 44.41% | 1201 | 33 | 92.66% | 89.27% | 1024 |
| D3 | 6 x 174 | 32 / 2 | 1,726 | 1,044 | 2,452,660 | 26.45% | 534 | 4 | 93.80% | 89.81% | 1344 |
| D4 | 3 x 162 | 32 / 2 | 1,823 | 589 | 3,190,925 | 42.19% | 261 | 3 | 92.02% | 86.28% | 2688 |
| D5 | 4 x 233 | 32 / 2 | 3,041 | 932 | 4,861,675 | 30.84% | 302 | 4 | 94.31% | 89.55% | 640 |
| D6 | 8 x 262 | 32 / 2 | 3,839 | 2,093 | 9,084,951 | 35.52% | 615 | 6 | 91.84% | 89.58% | 2496 |
| D7 | 7 x 238 | 32 / 2 | 4,838 | 1,664 | 8,634,398 | 35.15% | 1403 | 7 | 95.52% | 88.72% | 706 |
| D8 | 3 x 236 | 32 / 2 | 3,310 | 708 | 30,581,516 | 46.37% | 644 | 3 | 92.60% | 89.64% | 3136 |

The results of experiments involving regular and observation scan chains are summarized in Table 3.6. The entries regarding TC, the PC, and the average observability of scan cells have the same definition as those of Table 3.3. Recall that the table reports TC of 10 K pseudorandom test patterns and the number of tests needed to regain the 90% TC target. The column CB of Table 3.6 provides the total number of test data used to operate selection logic of maXpress per segment, including bits received by CF-decompressor. The latter number is equal to the product of the number of input channels and the number of scan shift cycles plus the size of the decompressor, i.e., variables uploaded during the decompressor initialization.

As can be seen, the TC of 10K pseudorandom test patterns is elevated in a very systematic fashion, whereas tests needed to regain the 90% TC target are shorter than those listed in Table 3.3. Also, the average observability of scan cells, including those not targeted by the X-masking scheme, remains very high. As a result, one may conclude that maXpress is capable of successfully working with the observation scan chains in such a way that their basic functionality is preserved despite of X states, and, as might be expected, given a PC, the use of observation scan increases the resultant TC, or alternatively it reduces PCs given a target coverage.

Table 3.6 Observation scan results – the group size $\left\lceil \sqrt{2s} \right\rceil$.

| | | The number of segments | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 4 | 8 | 10 | CB |
| D1 | TC % | 74.15 | 77.14 | 87.04 | 92.95 | 93.13 | 455 |
| | PC | > 16K | > 16K | 15,360 | 6,400 | 5,120 | |
| | OB+ % | 38.37 | 35.69 | 41.58 | 43.79 | 50.05 | |
| D2 | TC % | 78.60 | 82.42 | 87.51 | 92.31 | 92.56 | 470 |
| | PC | > 16K | >16K | 12,864 | 6,144 | 4,416 | |
| | OB+ % | 52.04 | 52.90 | 54.08 | 52.56 | 58.87 | |
| D3 | TC % | 88.55 | 90.00 | 92.28 | 93.53 | 93.60 | 535 |
| | PC | 10,688 | 10,240 | 5,120 | 3,264 | 3,072 | |
| | OB+ % | 85.10 | 84.82 | 85.93 | 88.39 | 89.26 | |
| D4 | TC % | 73.47 | 77.32 | 87.79 | 90.75 | 91.31 | 501 |
| | PC | > 16K | > 16K | 15,744 | 7,744 | 6,464 | |
| | OB+ % | 57.5 | 57.27 | 61.30 | 60.75 | 55.16 | |
| D5 | TC % | 83.45 | 87.58 | 91.03 | 92.07 | 92.14 | 681 |
| | PC | > 16K | 15,424 | 7,680 | 3,456 | 3,072 | |
| | OB+ % | 79.27 | 81.81 | 76.54 | 79.66 | 84.40 | |
| D6 | TC % | 67.23 | 72.44 | 84.05 | 91.10 | 91.33 | 737 |
| | PC | > 16K | > 16K | 15,488 | 5,952 | 4,416 | |
| | OB+ % | 52.01 | 51.24 | 53.62 | 60.89 | 58.49 | |
| D7 | TC % | 70.28 | 76.49 | 87.29 | 93.52 | 93.83 | 671 |
| | PC | > 16K | > 16K | 11,328 | 5,120 | 4,160 | |
| | OB+ % | 55.82 | 56.13 | 41.08 | 58.43 | 60.37 | |
| D8 | TC % | 73.16 | 75.78 | 85.51 | 91.23 | 92.09 | 735 |
| | PC | > 16K | > 16K | > 16K | 8,192 | 6,272 | |
| | OB+ % | 55.2 | 53.51 | 58.47 | 65.65 | 67.96 | |

# 4. Masking unknown values in deterministic in-system test

The following chapter introduces another user-tunable X-masking scheme [103], [122]. It works synergistically with an on-chip test compression logic by employing encoded test data to completely filter out unknown values that otherwise might reach a test response compactor such as a MISR or test result sticky-bits used by the on-chip compare framework. Moreover, methods to select control settings employed by the X-masking scheme to suppress X states in both per pattern and per cycle modes are discussed.

## 4.1 New scheme vs. maXpress

Although X-masking is regarded a mature area of research and development, the Moore's Law coupled with ever-increasing complexity in architectural design creates a need for new methods that could be used to handle X states in a cost-effective manner, especially with respect to the resultant hardware and optimal place and route. In response to these challenges, a new X-masking scheme, called maXpress, has been proposed in the previous chapter. It was designed to mask-out X values in logic BIST while (1) preserving the around 90% fault coverage (2) with the relatively small amount of test data. Consequently, maXpress works with relatively large clusters of pseudorandom patterns that share a reasonable amount of controls suppressing Xs per pattern and per cycle. A user-defined tunable structure of maXpress allows grouping of scan chains. As a result, it offers trade-offs between test logic complexity, the collateral damage caused by unpreventable masking of non-X values, the resultant test coverage, the test time, as well as test data needed to control X-masking itself.

However, as mentioned in Chapter 2.5, DIST requirements are much higher than those of logic BIST, and have a non-negligible footprint on any X-masking scheme serving DIST. This is confirmed here, as this chapter introduces a technique to filter X states out of responses produced within a DIST environment. It builds on certain architectural principles of maXpress to let the new scheme become reusable in the LBIST mode. Still, the new scheme is tightly integrated with the EDT technology [146] so that masking of Xs is controlled by EDT-encoded test data, i.e., data shared with test compression logic; this is done for each test pattern individually. Accordingly, it makes the entire approach compatible with sequential test data decompressors acting as PRPGs in test compression/LBIST hybrids. However, it implies changes in maXpress scan gaters and requires novel algorithms to select scan chains, pick the corresponding masks, integrate ATPG, reduce the amount of control data, and run fault crediting.

## 4.2 Compactor's circuitry

Fig. 4.1 shows a new test response compactor design. A modular X-masking logic (the grey area in the figure) outputs scan chain gating signals to block Xs originating at scan cells. The usual ATE input interface is replaced (if needed) with an on-chip memory storing compressed test patterns and control data, while test responses are channeled to a MISR through a group of XOR trees (spatial compactors).

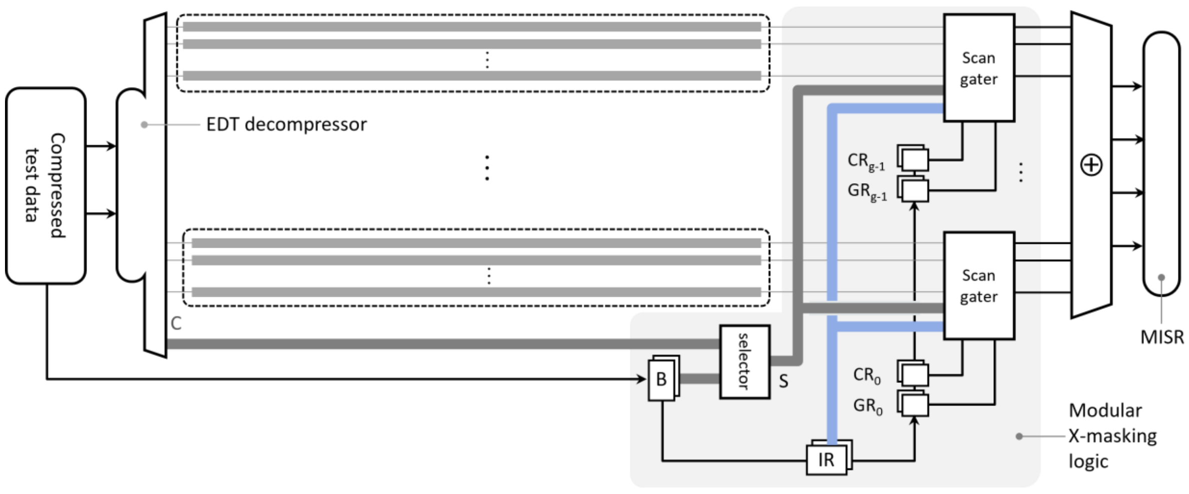


Figure 4.1 Test response compactor overview.

The actual blocking of test responses, captured by the scan chains, is carried out by *scan gaters*. These devices are located between the scan chain outputs and the MISR. One of the key structural features of the scheme is its equal grouping of scan chains such that each group is served by a dedicated scan gater. Let *n* be the number of scan chains served by a single scan gater. As can be seen (Fig. 4.2), it is comprised of 2*n* 2-input AND gates and *n*

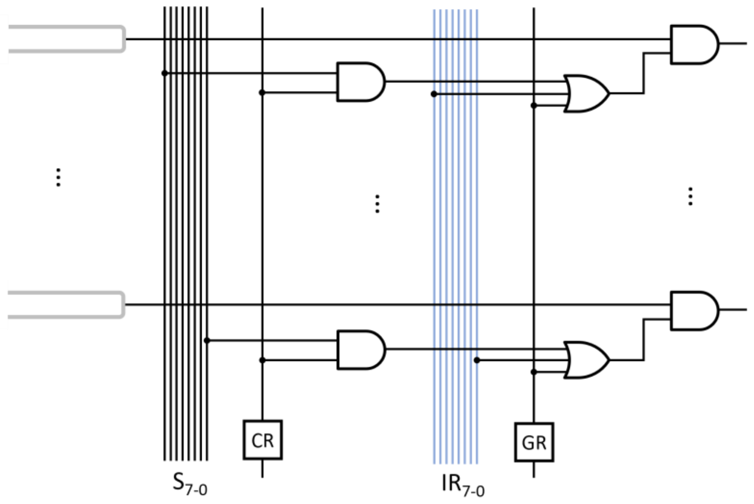


Figure 4.2 Scan gater serving *n* = 8 scan chains.

3-input OR gates. If *s* is the total number of scan chains, then the number of scan groups (and thus scan gaters) is given by $g = \lceil s / n \rceil$.

Scan gaters are individually controlled by the corresponding segments of a *g*-bit *configuration register* (CR) and a *g*-bit *group register* (GR). The first register enables – per pattern – a desired combination of scan gaters. The second one aims to increase observability of X-free groups, as detailed in Chapter 4.3. There is also an *n*-bit *index register* (IR) that is shared by all scan gaters, as shown in Fig. 4.1. This control allows deactivation of masking of *k*th scan chain from every group by setting a *k*th bit of IR to 1. Clearly, this should be done only if *k*th scan chain of every group is X-free. All registers described so far are daisy-chained with a 2*n*-bit register B (Fig. 4.3) to allow test data upload. As can be seen, all registers come with shadow counterparts to ensure updates at a proper time. As a result, reloads of the registers with new content can occur in parallel with a shift-in of data for the next pattern.

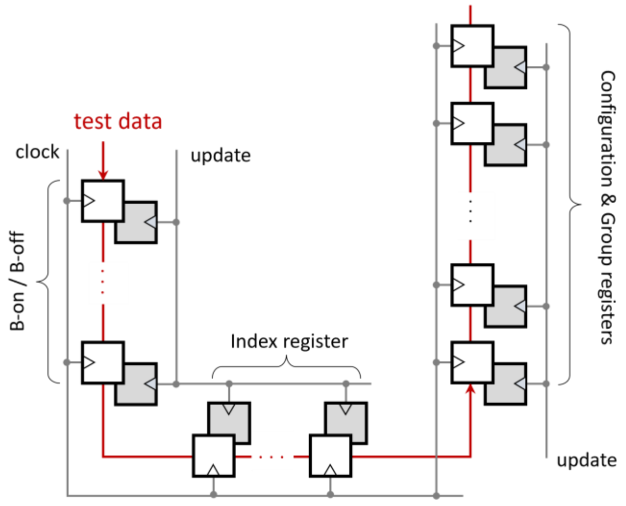


Figure 4.3 Feeding daisy-chained control registers.

Scan gaters enabled by CR receive masking data from a *selector* through a single bus **S** (in Fig. 4.2 the bus width is equal to 8). The selector consists of two *n*-bit registers B-off and B-on, and a simple combinational circuit that intakes primary inputs denoted as **C** (Fig. 4.4). Signals **C** are intended to filter test responses of corresponding scan chains in a *per cycle* mode unless values stored in registers B-off and B-on decide otherwise. These registers are reloaded once per pattern. Each scan chain is assigned two blockage bits. If *k*th bits of registers B-on and B-off are both set to 0, then scan chain *k* is blocked during the entire scan unload (the selector output $S_k$ is set to 0). Setting the *k*th bit of B-on to 1 makes scan chain *k* fully observable. Finally, deasserting the very same bit while asserting the corresponding bit of B-off allows us to mask scan chain *k* per cycle by means of **C** inputs. It is worth recalling here that all enabled scan gaters receive exactly the same controls. All control signals used by the proposed scheme are summarized in Table 4.1.

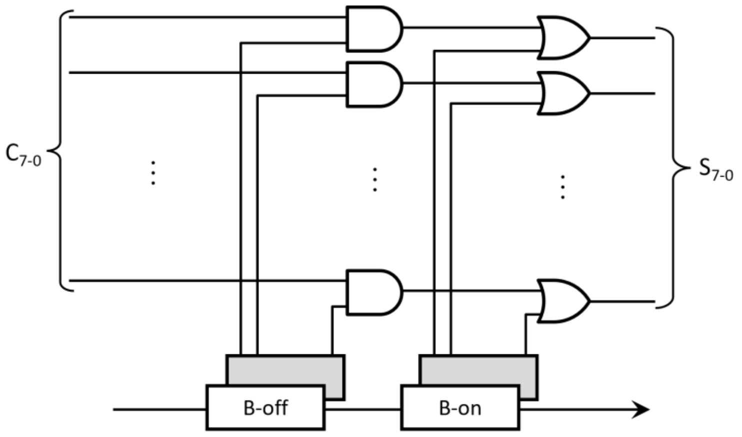


Figure 4.4 The selector block.

There is a large body of experimental evidence that per-cycle control settings are usually used for just a few scan chains. Therefore, data occurring on inputs **C** can be EDT-encoded and delivered to the selector by the EDT on-chip decompressor. This is a key finding for the modus operandi of the test response compactor shown in Fig. 4.1. For further details on how to master steps needed to make EDT-based compression part of the X-masking see the next chapters.

Table 4.1 Control signals.

| Name | Purpose | Update |
|---|---|---|
| GR | Select *groups* of scan chains to be observed | Per pattern |
| IR | Select *indexed* scan chains to be observed | |
| CR | Select *groups* of scan chains to be masked by S | |
| B | Mask/observe indexed scan chains in CR-selected *groups* | |
| C | Mask/observe scan cells in CR-selected *groups* | Per cycle |
| S | Controls shared by CR-selected *groups* | |

## 4.3 Mask generation

Selection of masking (or gating) signals plays a key role in the proposed test scheme. With three levels of masking, finding values used to either block or observe groups of scan chains, individual chains, and finally individual scan cells, is a nontrivial process. This chapter presents an automated method that aims at assuring observability of scan cells which are unique fault propagation sites while reducing an overmasking of non-X values.

We begin by setting the group register GR, and then, orthogonally, the index register IR. These registers are dominators of the entire masking logic, as can be seen in Fig. 4.2. The group register GR allows one to observe those bundles of scan chains that are entirely X-free. Furthermore, if there are X-free scan chains in all groups, and these chains have the same index, then the index register IR unlocks them by sharing the corresponding bit with all groups. Faults that propagate to scan chains GR- or IR-unlocked are detected by default (except an unlikely event of aliasing), and therefore they are not processed any further. In particular, they do not play a part in generation of weights, as described below, thus raising the chances of other faults to be observed and detected. Moreover, the use of GR and IR increases the observability of cells that do not capture target faults. This, in turn, may increase the likelihood of detecting unmodeled faults propagating to scan cells that are not observed on purpose.

The next paragraphs will use the following notion. A fault detected by *q* test patterns will be designated as *q*-D, e.g., 1-D, 2-D, 3-D, etc. In particular, faults detected by just a single test pattern (1-D) are often referred to as essential faults, and they are the most crucial ATPG targets.

Scan chains not served by GR and IR are put through an additional mask generation process that resembles the approach presented in Chapter 3, although masks are determined individually *per pattern* rather than a group of patterns. Consider the example shown in Fig. 4.5. Here, there are ten scan chains, each comprising eight scan cells. These scan chains are split into two groups. Let us also assume that there are three test patterns $P_1$, $P_2$, and $P_3$. The following description will demonstrate how to obtain weights for a test response corresponding to the first pattern ($P_1$). Propagation sites of Xs (X-cells) and faults are indicated by red and labeled circles, respectively. While certain faults are detected (and thus observed) more than once, special attention should be paid to faults detected by just a single test pattern, further referred to as 1D faults. In principle, such faults are the most important ATPG targets, and hence to guarantee their observability is of primary concern. As faults from A to G belong to the class of 1D faults (they are only detected by $P_1$), all their propagation sites are assigned a weight equal to 1, even if some of them propagate to multiple sites, such as faults B and F. Following this convention, X cells get the 0 weight. Faults H, J, K, and L (blue circles) are

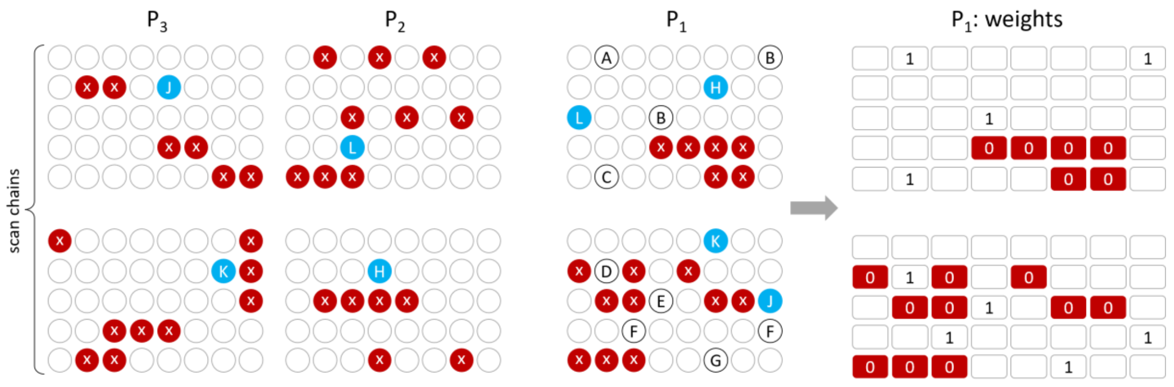


Figure 4.5 Generation of weights for 1D faults.

also detected by patterns $P_2$ and $P_3$. Hence, their propagation sites are not assigned any weight as far as the response corresponding to pattern $P_1$ is concerned. However, if they cannot be observed after applying masking signals for pattern $P_1$, faults L and H will become 1D faults in a test response corresponding to $P_2$, while faults K and J will get the same status in a test response obtained after applying pattern $P_3$. The same rule applies to all $q$-D faults, $q > 1$: in the worst case (they are masked due to Xs in $q - 1$ test responses) each of them will eventually get the status 1-D and will be subjected to the weight generation process.

The weights associated with successive observation sites are subsequently used to superpose groups of test responses, as illustrated in Fig. 4.6. X-free scan chains with fault observation sites as well as chains with no observation sites (D-free) but having Xs are enabled/disabled using a proper combination of B-on/B-off values. Back to Fig. 4.6, the first scan chain of every group is to be observed by asserting the first bit of the B-on register, whereas the last scan chain of every group will be blocked after de-asserting the last bits of B-off and B-on registers. The remaining scan chains receive per-cycle data from the selector (see the next chapter): Xs are masked with the value of 0, and faults are observed with the masking values set to 1. It is worth noting that some fault observation sites may overlap with certain X-cells when superposing groups of test responses. As masking of X-cells is mandatory, it will inevitably result in over-masking of faults. The mask selection has to take account of this phenomenon.

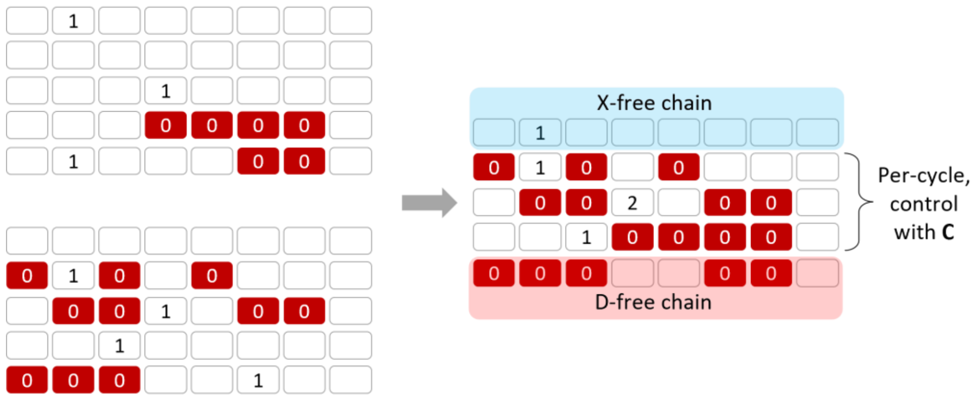


Figure 4.6 Superposition of groups.

Selection of controls is an iterative greedy process as outlined in Fig. 4.7 for a single pattern. It uses a $g$-bit binary vector $\mathbf{H} = [h_{g-1} \dots h_1 h_0]$ representing the configuration register CR, i.e., $h_k = 1$, if $k$th stage of the register is set to 1 (recall that $g$ is the number of groups). Essentially, after random initialization of **H**, the method inverts successive bits of **H** and check the resultant masking solution, keeping the best one, i.e., a group selection that yields the largest population of 1D faults which are observed. It then becomes a starting point for the next iteration. Typically, N = 3 repetitions of the outer loop (including random initialization of **H**) suffice to find the best masking configuration. If there are several masking configurations with the same number of observable 1D faults, the number of other faults observed fortuitously is used to break a tie. Having determined the group selection for a given pattern, a list of 1D faults can be updated. Furthermore, faults whose observability could not be secured have their status changed from $q$D to ($q$–1)D, with 1D faults becoming new additions to their list.

```
Maximum sum M ← 0
while the number of iterations is less than N
    select randomly initial state of H
    Temporary maximum sum T ← 0
    do
        for each bit in H
            set a given bit to its opposite value
            compute sum S of weights
            if S > T then T ← S
        set H to the value of the best vector resulting in T
    while T changes its value
    if T > M then M ←T
update list of 1D faults
```

Figure 4.7 Finding masks.

## 4.4 Encoding of masking cubes

It has been demonstrated earlier how to determine the per-pattern content of the configuration, group, index as well as B-off and B-on registers. The control signals that can be contrasted with these settings are masking values provided through the selector inputs **C** in a per-cycle regime (Fig. 4.4). As shown in Chapter 4.2, these masking signals are to be EDT-encoded and delivered by the main EDT decompressor. Hence, encoding of a given test pattern $p$ is *preceded* by an attempt to encode a masking cube associated with the previous pattern $p$ – 1. This process can be characterized as follows.

In principle, a two-dimensional masking cube consists of as many cycles as the scan chains length while the number of rows matches the number of scan chains per group. Its entries are set to 0 or 1, if one needs to mask Xs or enable fault observation sites, respectively. These rules do not apply to the entries that are observed or blocked by B-on and B-off registers. All the remaining entries are considered don't cares (–). Back to Fig. 4.6, a masking cube corresponding to the superposed pattern (the right-hand side of the figure) would have the following form:

$$\begin{array}{cccccccc}
- & - & - & - & - & - & - & - \\
0 & 1 & 0 & - & 0 & - & - & - \\
- & 0 & 0 & 1 & - & 0 & 0 & - \\
- & - & 1 & 0 & 0 & 0 & 0 & - \\
- & - & - & - & - & - & - & -
\end{array}$$

Starting with $q = 1$, the process of forming a masking cube (see also Fig. 4.8) begins with the creation of a list **L** comprising all sites observing *q*D faults. Recall that they were obtained as a result of group superposition (Fig. 4.6). The list is sorted in descending order, i.e., an entry with the largest count of *q*D faults comes first. After filling a masking cube with 0s corresponding to all X-cells (this initial step is mandatory to ensure that all X states will be blocked), we iteratively pick the first item on list **L**, set the corresponding entry of the masking cube to 1, delete that item from the list, and try to encode the resultant cube. If the encoding try fails, then a newly added 1 is removed from the masking cube. It is worth noting that it need not necessarily preclude detection of certain faults provided they propagate to other observation sites. Moreover, assuring observation of a given site results in dropping all faults that propagate to it. The above steps are carried out until list **L** becomes empty. Once all test patterns have their masks encoded with respect to *q*D faults, the process repeats for the next (incremented) value of *q* until all faults are processed. This iterative approach optimizes the usage of masking variables with 1D faults being targeted in the first round. Clearly, other sites that have been successfully encoded in every step may enable fortuitus observability of other faults that remain to be tackled.

```
q ← 1
while there are any target faults
    create list L of sites observing qD faults
    sort L in descending order of the number of qD faults
    pick the first item S from L
    add S to the masking cube
    if encoding fails then remove S from the masking cube
    else drop all faults in S from the fault list
    remove S from L
  q ← q + 1
```

Figure 4.8 Encoding of a masking cube.

Borrowing input variables to encode masking data may reduce the EDT-based encoding capabilities, but typically the X-masking process requires a very small fraction of the total number of variables, and hence it does not compromise the quality of compression of the actual test patterns. This is clearly confirmed in Fig. 4.9. It plots experimental data showing how masking cubes consume EDT input variables as a function of test application time (these curves represent a moving average to smooth out the actual data points). The experiments were run for four industrial designs (for further details regarding these designs see Chapter 4.5). As can be seen, the use of EDT variables tends to decrease as tests proceed. At the end, design D9 needed only 5% of the total test data budget, whereas designs D1 and D4 used just a few bits to encode per-cycle masking signals. More importantly, however, what was needed in all experiments to encode masking data at any time was just a small fraction of input variables injected through a single EDT input channel.

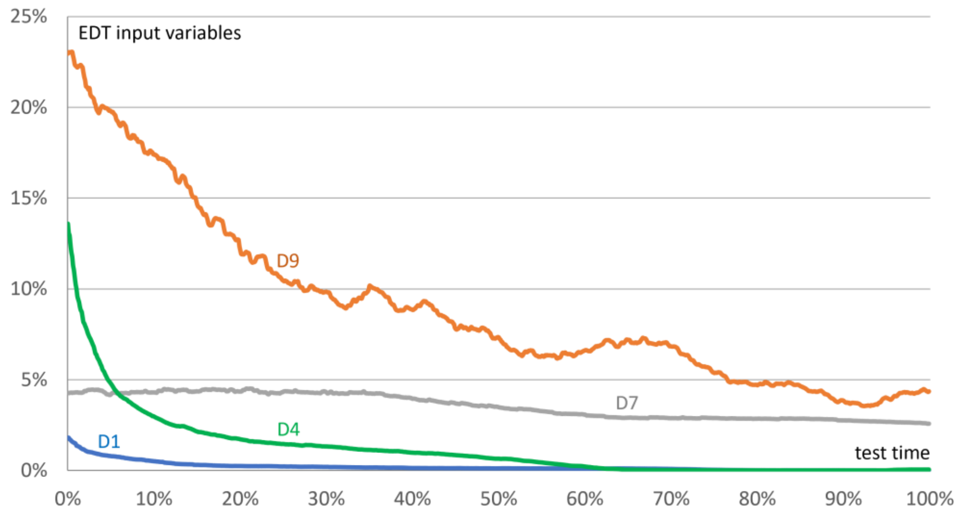


Figure 4.9 Test data usage by masking cubes.

In the unlikely event of a decreased degree of test cube merging due to X-masking, one will just observe a slight increase in a pattern count. It appears that securing observability of a given 1D fault in the first place is less expensive in terms of test data usage than encoding a new test cube targeting that fault again. It is also worth recalling that faults propagating to scan chains selected by either IR or GR registers are not subject to mask encoding. It may substantially reduce demands for seed variables. Table 4.2 presents how many faults, out of all faults detected so far and reported in the column "Total", are captured in scan chains selected by IR and GR registers during the first 64 and then 640 test patterns deployed in ten industrial designs. In fact, more than 50% of faults can be detected this way when applying the very first test patterns. Typically, these are *q*D faults, where $q > 1$. In other words, a majority of faults that require a per-cycle mask encoding and are detected by tail-end test patterns are 1D faults.

Table 4.2 Faults observed in scan chains selected by IR and GR.

| | After 64 patterns | | | After 640 patterns | | |
|---|---|---|---|---|---|---|
| | Total | IR&GR | % | Total | IR&GR | % |
| D1 | 792846 | 699982 | 88.29% | 1181133 | 973500 | 82.42% |
| D2 | 2021033 | 2016535 | 99.78% | 2647322 | 2613605 | 98.73% |
| D3 | 2165261 | 2160930 | 99.80% | 2591442 | 2577347 | 99.46% |
| D4 | 1531689 | 1072445 | 70.02% | 2113270 | 1368517 | 64.76% |
| D5 | 1593889 | 1401699 | 87.94% | 2658616 | 2128851 | 80.07% |
| D6 | 1207023 | 1198191 | 99.27% | 1376839 | 1335092 | 96.97% |
| D7 | 5212999 | 4960110 | 95.15% | 7363481 | 6769248 | 91.93% |
| D8 | 157352 | 138739 | 88.17% | 188675 | 153555 | 81.39% |
| D9 | 2672367 | 2137507 | 79.99% | 3383130 | 2357365 | 69.68% |
| D10 | 2318705 | 1737063 | 74.92% | 3084714 | 2074486 | 67.25% |

## 4.5 Experimental results

Several experiments outlined below were performed with 10 large industrial cores having all components of the proposed compactor on a chip. Table 4.3 lists major characteristics of the examined test cases: the number of gates, the number of scan cells, and the scan architecture. Furthermore, Table 4.3 reports the following metrics:

- the number of stuck-at faults,
- the number of EDT input channels and the size of the EDT decompressor,
- the total number of scan cells that capture X states (X-cells) across all test patterns,
- the total number of scan chains that capture X states (X-chains) across all test patterns,
- the reference test coverage recorded for a test setup deploying the EDT default X-tolerant test response compactor X-Press [149].

The number of groups (Fig. 4.1) used in the experiments is equal to $\lceil \sqrt{2s} \rceil$, where $s$ is the number of scan chains; such grouping minimizes the number of memory elements used to store the control data, as shown in Chapter 3.3. Designs reported in Table 4.3 feature a certain spectrum of X-fill rates represented by the number of X-cells ranging from 223 (D3) – it corresponds to 0.12% of the entire population of scan cells – up to 9,668 (6.52%) for D10. Furthermore, the amount of scan chains that capture any unknown values may also impact the final results. For example, X states propagate to a few scan chains in several designs in a uniform manner across all test patterns reaching over 54% of all scan chains (D10), whereas there are only a few percent of scan chains with some accumulation of X states in design D2, D3, and D7 where the vast majority of scans (more than 96%) have no unknown states at all.

Table 4.3 Circuit characteristics.

| | Gates | Scan cells | Scan | Stuck-at faults | EDT inputs / size | X-cells | X-chains | Test coverage [%] |
|---|---|---|---|---|---|---|---|---|
| D1 | 1.02M | 35.6K | 144 × 249 | 2,721,968 | 8 / 32 | 2,213 | 56 | 97.89 |
| D2 | 2.47M | 149.4K | 1,400 × 374 | 5,933,388 | 2 / 46 | 837 | 36 | 99.71 |
| D3 | 2.43M | 185K | 500 × 371 | 6,354,467 | 2 / 60 | 223 | 16 | 99.84 |
| D4 | 1.21M | 72.3K | 381 × 190 | 3,812,564 | 16 / 64 | 7,185 | 148 | 96.40 |
| D5 | 2.09M | 145.1K | 420 × 346 | 5,610,954 | 4 / 46 | 6,906 | 101 | 98.54 |
| D6 | 1.18M | 97.8K | 300 × 327 | 4,251,354 | 2 / 37 | 472 | 22 | 97.65 |
| D7 | 7.86M | 428.7K | 857 × 502 | 8,357,022 | 8 / 32 | 1,981 | 10 | 92.65 |
| D8 | 0.22M | 12.6K | 122 × 138 | 188,486 | 4 / 32 | 375 | 9 | 98.97 |
| D9 | 2.49M | 173.7K | 114 x 1964 | 3,688,965 | 4 / 32 | 5,708 | 39 | 91.23 |
| D10 | 2.14M | 148.1K | 70 x 2579 | 6,898,121 | 4 / 32 | 9,668 | 38 | 98.20 |

Table 4.4 summarizes the key experimental results. It lists the following outcomes:

- the total amount of (static) control data used to feed (per pattern) the configuration, group, index as well as B-off and B-on registers; as can be seen, the static control data is a small fraction of the overall data volume (see the last but one column of the table) used by the proposed solution, i.e., it takes a bit more than 2% for design D6, it reaches 15% for D3, while the average value over the reported designs equals 7.54%,
- the fraction of EDT seed variables needed per pattern to encode the masking cubes; this figure of merit is further represented by three numbers: lower (different than 0) and upper extremes, as well as the average value; for example, the value of 3.49 in the column average indicates that only 3.49% of all seed variables deployed by the EDT-based compression were used, on the average, to encode the masking cubes handling test responses in the per-cycle mode; as one may expect, the number of EDT seed variables needed to encode the masking cubes is typically a negligible fraction of the total number of variables employed to encode successive test patterns with only one exception of design D10 (here the ratio is above 9%),
- the effective test coverage achievable with the proposed scheme based on a MISR-produced signature.

Table 4.4 Experimental results.

| | Static control data [Mb] | Dynamic control data per pattern [% of EDT seed variables] | | | Test coverage [%] | Pattern count | | Data volume [Mb] | | Reduction ratio |
|---|---|---|---|---|---|---|---|---|---|---|
| | | min | max | average | | X-Press | DIST | X-Press | DIST | |
| D1 | 0.145 | 0.05 | 16.30 | 0.88 | 97.89 | 3,040 | 2,112 | 17.5806 | 4.1572 | 4.23x |
| D2 | 1.6885 | 0.13 | 40.13 | 0.57 | 99.71 | 15,797 | 15,232 | 34.7705 | 12.4927 | 2.78x |
| D3 | 1.3438 | 0.13 | 43.92 | 0.09 | 99.84 | 11,897 | 11,008 | 26.0047 | 9.1333 | 2.85x |
| D4 | 0.2051 | 0.03 | 76.32 | 2.07 | 96.40 | 1,920 | 1,920 | 17.0801 | 5.7715 | 2.96x |
| D5 | 0.9841 | 0.02 | 65.35 | 1.06 | 98.54 | 11,328 | 8,896 | 45.7625 | 12.7258 | 3.60x |
| D6 | 2.4225 | 0.14 | 54.66 | 0.15 | 97.65 | 22,976 | 22,680 | 354.793 | 115.5872 | 3.07x |
| D7 | 0.8738 | 0.02 | 25.52 | 3.49 | 92.65 | 5,888 | 5,454 | 68.416 | 21.7624 | 3.14x |
| D8 | 0.0459 | 0.17 | 9.76 | 0.03 | 98.97 | 764 | 752 | 1.2299 | 0.4418 | 2.78x |
| D9 | 0.375 | 0.01 | 81.11 | 9.68 | 91.23 | 6,015 | 6,144 | 135.3779 | 46.4063 | 2.92x |
| D10 | 0.5619 | 0.01 | 61.74 | 0.62 | 98.20 | 3,903 | 3,876 | 22.5416 | 7.9843 | 2.82x |

In addition to the metrics listed above, the last section of Table 4.4 provides a comparison between a reference EDT-based *on-chip compare* scheme and the solution presented in this chapter. The reference scheme deploys the X-Press test response compactor of [149] with the same number of outputs as the number of EDT inputs shown in Table 4.3. The total volume of data used by X-Press contains its own control settings and the reference data. The reference volume V is equal to

$$V = 2 \times o \times p \times L, \tag{4.1}$$

where $o$ is the number of X-Press outputs, $p$ is the number of patterns, and L is the size of the longest scan chain. This quantity has to be *doubled* to account for extra bits needed to differentiate between X and non-X results. Consequently, the last five columns of the table display respectively:

- the X-Press-based pattern count,
- the new pattern count,
- the X-Press data volume, including the EDT seed variables (in megabits),
- the new data volume; this quantity includes primarily the number of seed variables obtained by multiplying the number of test patterns, the number of EDT input channels, and the size of scan chains plus initialization cycles, the remaining tiny part is shown in the first column reporting the static control data,
- a ratio between the total volume of test data used by the reference X-Press-based on-chip compare technique and the method introduced in this chapter.

As can be easily verified, the proposed X-masking scheme does not compromise the test quality – test coverage remains unaffected in all test cases. Similarly, the resultant pattern counts of the new scheme remain virtually the same as those of the X-Press based approach. It is worth recalling that X-Press [149] treated here as the reference is not even required to mask all unknown states as opposed to the proposed scheme. In fact, the pattern counts get even decreased for three examined circuits. Other designs observe a slight increase in their pattern counts. This phenomenon is attributed to a distribution of X states which in order to be masked intercept (occasionally) a non-negligible fraction of EDT seed variables. This, in turn, reduces the EDT-based encoding capacity and thus elevates the pattern counts. Nevertheless, the new scheme reduces the total test data volume in all test cases by the average factor of 3.12x relative to the on-chip compare solution.

Another pragmatic metric used to characterize a new DFT solution is its test logic silicon real estate. As shown in Chapter 4.2, the new X-masking scheme requires three logic gates per scan chain. Additional gates are used to implement the selector; here two logic gates are needed per the selector output – their total number is equal to the number of scan chains placed in a single group. The total gate count equals $3s + g \approx 3s$. Also, two flips-flops (CR and GR) are employed per group (or a scan gater) altogether with their shadow counterparts. Furthermore, the index register, registers B-off, B-on, and their shadows require, in total, $6n$ flip-flops, where $n$ is the number of scan chains served by a single scan gater. Eventually, the total number of flip-flops is given by $4g + 6n \approx 7g$. It can be verified that if the number $s$ of scan chains is greater than or equal to 100, then the flip-flop count becomes smaller than $s$.

Consider, as an example, design D5 that features 2.09M gates and *s* = 420 scan chains, with the longest chain comprising 346 memory elements (see Table 4.3). In total, this design employs 145,177 scan cells. Its scan chains are evenly divided into 29 groups, with almost all groups comprising 15 chains. D5 requires additional 206 flip-flops and 1290 gates to implement the proposed X-masking logic. Hence, the fraction of extra flip-flops is equal to 0.14%, while the total area overhead incurred by the new scheme (assuming that a single flip-flop is equivalent to 6 gates) amounts to 0.12%. Similar results can be obtained for the remaining designs, and they are listed in Table 4.5.

Table 4.5 Hardware footprint of a new scheme.

| | #Groups | #chains per groups | Extra gates | Extra flip-flops | Total area [%] |
|---|---|---|---|---|---|
| D1 | 17 | 9 | 449 | 122 | 0.116 |
| D2 | 53 | 27 | 4253 | 374 | 0.263 |
| D3 | 32 | 16 | 1532 | 224 | 0.118 |
| D4 | 28 | 14 | 1171 | 196 | 0.194 |
| D5 | 29 | 15 | 1289 | 206 | 0.121 |
| D6 | 25 | 13 | 925 | 178 | 0.169 |
| D7 | 42 | 21 | 2613 | 294 | 0.056 |
| D8 | 16 | 8 | 382 | 112 | 0.479 |
| D9 | 16 | 8 | 358 | 112 | 0.041 |
| D10 | 12 | 6 | 222 | 84 | 0.034 |

# 5. Hardware security and IC tests

As shown in the previous chapters, DIST applying deterministic test patterns and using test response compaction protected by the new X-masking schemes is well positioned to deliver high quality manufacturing tests for large SoC designs. However, the very same test framework may provide unrestricted access to internal states of a device-under-test. Thus, it opens a backdoor for security threats such as IP theft/piracy, reverse engineering, counterfeiting, tampering, or IC overproduction. This scenario is similar to other scan-based test schemes that can make a circuit-under-test potentially vulnerable to various forms of attacks trying to retrieve or modify sensitive data and assets [181]. Although the presence of on-chip test compression and encoded test data make a circuit more resistant to scan attacks, test compression facilities are not as effective countermeasures to scan-launched attacks [46] as one might expect. Notwithstanding the fact that an SSN forms yet another defense line, it remains essential to apply access restrictions and to secure test infrastructure to prevent leakage of any secret information while tests are carried out. Consequently, the second part of the thesis is devoted to design and analysis of several new security primitives that can eventually establish a lightweight hardware root of trust guarding a variety of DFT ecosystems. It begins with a brief review of a large volume of published studies describing techniques that address security issues caused by IC testing, and then move on with the presentation of new security solutions.

## 5.1 Design for test vs. security concerns

A wide range of solutions have been proposed so far to secure test circuitry and test access ports in general, and scan chains in particular, with the aim of making it more difficult to launch scan-based attacks. One of the first countermeasures was randomly inserting inverters between scan cells [164]. As a result, test patterns and test responses are transformed during the shift-in and shift-out test phases, respectively. This simple approach, however, can be easily broken by resetting a chip and shifting data out in the test mode. Positions of 1s in the resultant response reveal locations of the corresponding inverters. A solution immune to such reset-based attacks was presented in [8]. As shown in Fig. 5.1a, 3-input XOR gates are used to combine a given scan cell input with the outputs of two successive flip-flops. As shown in [7], this approach is not secure, either. In addition to a method unveiling locations of XOR gates, the authors of [7] presented another technique to secure scan chains. To eliminate purely deterministic approach and to add randomness to the XOR-based security mechanism, they used a physical unclonable function (PUF). Fig 5.1b shows a PUF serving as a source of multiplexer controls. As a result, signals from either the preceding scan cell or the XOR gate are selected in a pseudorandom fashion. While this change increases the overall security of DFT logic, changes made to scan chains are still static – modifications are test independent

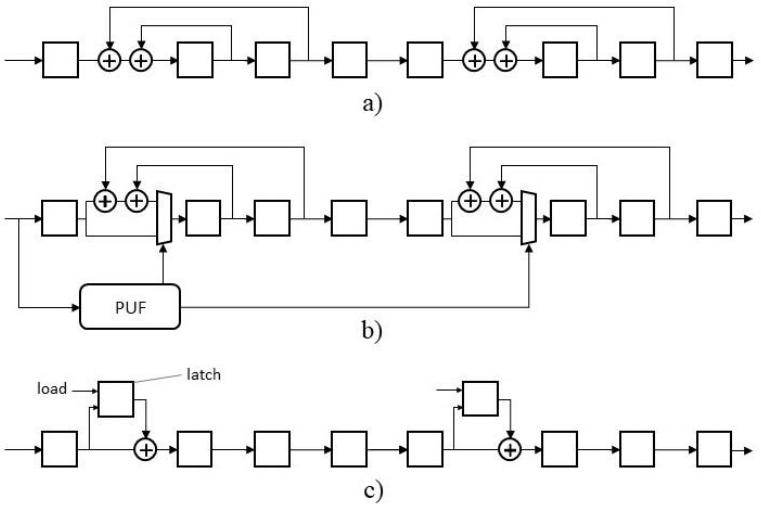


Figure 5.1 (a) Double feedback XOR scan [8],
(b) Secure scan with PUF-based feedback selection [7],
(c) Dynamically modified scan structure employing state dependent scan flip flop [5].

as they are based on a known PUF response. A secure scan architecture where scan data is changed dynamically was presented in [5]. To achieve this functionality, randomly selected scan cells are converted into state-dependent scan flip-flops (SDSFF). With the addition of a latch and an XOR gate, the SDSFF value is determined by both flip-flop and latch values. As shown in Fig. 5.1.c, an update of the latch value depends on the load signal. The authors recommended asserting the load signal every *n* clock cycles, with *n* set individually for every circuit to further reduce chances of a successful attack.

Other secure scan architectures deployed key-based scrambling mechanisms [30], [78], [97], [153], or additional registers to separate critical and regular data [201], regrouped scan chains into sets of smaller sub-chains [98], or took advantage of a combinational ATPG with a secure partial scan [32], [84]. However, a secure scan architecture cannot be considered safe when an attacker can get design netlists through *reverse engineering* which has to be, therefore, considered a real threat [19], [90]. Knowing the locations of additional logic like XOR feedbacks or latches, one can easily access scan data by virtue of, for example, a Boolean satisfiability based [170] attack. Therefore, to increase chip security, it is mandatory to employ techniques that guarantee access to a test infrastructure only to authorized users. One of the techniques presented in [129] introduces two additional instructions to the JTAG TAP controller. By invoking one of these instructions, one can either block or gain access to the test infrastructure, provided a correct password is used. Similarly, the solution presented in [36] modifies the IEEE 1500 test wrapper by employing an additional LFSR. Users can unlock the test wrapper with the proper combination of an LFSR's seed and a golden key. The key corresponds to the final state of the LFSR; thus the seed/key combination can only be generated, if one knows an LFSR characteristic polynomial. Clearly, this knowledge is only granted to legitimate users.

The password-based solutions presented in [36] and [129] relies on the fact that a secret password or an LFSR polynomial is known only to authorized users. However, if attackers could obtain this data, all the countermeasures become ineffective. Fortunately, the security of test infrastructure can be improved by adapting challenge-response communication protocols. Solutions based on symmetrical cryptography employ, for example, SHA-256-based protection [37]. Here, a device generates a unique, random challenge and sends it to a user. The challenge is then combined with the secret key and hashed with the SHA-256. Both hash values are then compared on the chip – they can only match if the user and the design store the same secret key. In the PUF-based protection [45], the user must store a PUF challenge-response pairs (CRPs) instead of a secret key. Here, the challenge sent by a device is a desired, random distance between two PUF responses. The user looks through all the stored CRPs to find a proper pair of challenges and sends it back to the device. Finally, access to a test infrastructure is granted only if the obtained distance is the same as the expected one. However, the PUF-based approach may also need to store individual CRP databases for each design and to read out PUF responses during the manufacturing phase. What is more, a SHA-256 hardware implementation may be problematic, especially within small devices with limited silicon area resources. Consequently, the challenge-response authentication is typically carried out by a device implementing *lightweight* crypto hash functions.

## 5.2 Lightweight cryptographic hash functions

In recent years, the number of publications devoted to lightweight crypto hashing has been increasing steadily in volume and importance as hash functions became key hardware security primitives. The existing lightweight cryptographic hash functions can be classified into a few groups determined by a construction method [191]. These constructions include (a) the Merkle-Damgård scheme [44], [113], (b) the sponge construction [15] with the Keccak hash function [16] – its most prominent, although not lightweight, instance and the winner of the NIST SHA-3 secure hash standard competition, (c) block-cipher-based solutions such as the Davies–Meyer hash functions [138] or functions deploying nonbinary error-correcting codes [94], and finally (d) methods which are based on cellular automata [75] or (e) deterministic chaotic finite state machines [2].

Lightweight cryptographic hash function, while designed to be compact, fast, and self-testable within resource-constrained devices, must be still relatively immune to brute force and cryptanalytic attacks. Typically, it is expected that a cryptographic hash function H has the following properties related to the brute force attacks:

- *preimage resistance* (one way): for virtually every output $z$, it is computationally infeasible to determine input $x$ hashing to that output, i.e., $H(x) = z$,
- *second-preimage resistance*: it is computationally infeasible to determine another (distinct) input $y$ hashing to the same output as any given input $x$, i.e., $H(x) = H(y)$,

- *collision resistance*: it is computationally infeasible to find two colliding inputs $x$ and $y \neq x$ such that $H(x) = H(y)$.

Each resistance is evaluated as an $n$-bit security, meaning that to conduct a successful brute force attack, one should perform (on average) $2^n$ operations. For a hash function of a size $m$, the best possible preimage resistance and second-preimage resistance $n = m$. For example, a successful preimage and second-preimage attack on a 256-bit hash output would require a $2^{256}$ hash generations (assuming highest possible resistances). On the other hand, an ideal collision resistance $n = m/2$, as is indicated by a birthday paradox. Table 5.1 presents the resistance of the selected lightweight cryptographic hash functions, where column "Size" reports the hash value length.

Table 5.1 Security parameters of selected lightweight cryptographic hash functions.

| Hash function | | Resistance | | |
|---|---|---|---|---|
| Name | Size | Preimage | 2nd Preimage | Collision |
| ARMADILLO | 160 | 160 | 160 | 80 |
| | 256 | 256 | 256 | 128 |
| ASCON | 256 | 128 | 128 | 128 |
| PHOTON | 160 | 124 | 64 | 64 |
| | 256 | 224 | 128 | 128 |
| DM-PRESENT | 64 | 64 | none | none |
| H-PRESENT | 128 | 128 | none | none |
| QUARK | 256 | 224 | 112 | 112 |
| SLISCP | 160 | 128 | 80 | 80 |
| SPONGENT | 128 | 120 | 64 | 64 |
| SPONGENT | 256 | 240 | 128 | 128 |

Metrics shown above describe the resistance to attacks based on brute-force methods. More sophisticated techniques are based on the cryptanalytical approach, i.e., identifying weak points of hash functions in their structures and algorithms. Cryptanalytical attacks currently known are based on differential, integral, algebraic, or linear analysis. Other popular methods include cube, slide, or rebound attacks as well as zero-sum, rotational, or meet-in-the-middle distinguishers and truncated or impossible differentials. Due to architectural differences between targeted functions, each cryptanalysis is usually dedicated to a single hashing algorithm. An overview and references to the cryptanalytical methods for lightweight cryptographic hash functions can be found in [191].

It is widely agreeable that a proper hash function matching desired hardware and safety requirements is an indispensable component required to implement a secure challenge-response procedure. However, to increase the overall security, the authentication phase may be combined with the encryption/decryption of test data by using *stream ciphers*.

## 5.3 Hardware stream ciphers

Stream ciphers (SCs) are other key security primitives that can be successfully used in the area of VLSI test. Their mission is to encrypt and decrypt streams of test data by combining them with the secret, cipher-produced, cryptographically secure pseudorandom keystreams. Since the role of a combiner is typically assumed by the exclusive-or operations, binary additive ciphers [93] make up the vast majority of actually implemented SCs.

As documented by a large volume of scholarly literature on hardware SCs, many SC designs are based on nonlinearly filtered sequences produced by LFSRs. Generators that were proposed by Geffe, Jennings, Beth and Piper, Rueppel, Rao, Briier, Massey and Rueppel, or Chambers and Gollmann belong to this category; the corresponding details can be found in [161]. Other schemes include the shrinking generator [39], WG [61], [70], the self-shrinking generator [112], the reconfigurable feedback shift register [207], the Toeplitz-hash-based ciphers [48], SNOW [60], as well as XPD, Nanoteq, Rambutan, M, and Gifford's algorithms [161]. Some are based on congruential additive generators (Fish, Pike, Mush [161]), or, like the Blum-Blum-Shub generator [17], reuse certain concepts employed in public key cryptography. More recent schemes integrate LFSRs with sequential circuits having nonlinear feedback networks. Such solutions can be found among the finalists of the eSTREAM competition funded by the European Union. Their list includes Decim, Edon80, F-FCSR, Grain, Mickey, Moustique, Pomaranch, and Trivium. Detailed descriptions of those significant milestones are available in [154]. It is worth noting that the Grain cipher has evolved into a family of solutions including small-state SCs such as Sprout, Fruit, Plantlet, or Lizard [89]. The same has been observed for the Trivium-like designs.

It appears that LFSRs can be replaced with nonlinear feedback shift registers (NLFSRs), as implemented in Achterbahn-128 [68]. Interestingly, some SCs have never been officially disclosed. Nevertheless, their details eventually became publicly available. The SCs E0 (developed for Bluetooth technology), A5/1 (used by GSM), and RC4 (deployed in the 802.11 wireless LAN standard) can serve here as examples. More advanced designs may use special forms of finite state machines, as done in Stanislaw [65]. Another group of SCs belonging to this category are designs deploying certain concepts used by cryptographic hash functions such as Keccak. For example, the Ascon family of SCs is based on a sponge framework [55]. Since many SCs share similar building structures with hash functions, methods used for SCs cryptanalysis resemble those mentioned earlier. The comprehensive surveys of the most representative and state-of-the-art solutions in the SC domain can be found in the review papers such as [85], [135], [204], [205]. They also discuss SC resilience against various forms of attacks and highlight schemes that can be used even in compact devices with limited computing resources.

## 5.4 Hardware root of trust

Cryptographic hash functions and SCs play a vital role in shaping high-end hardware roots of trust (RoT) – foundations on which secure operations of digital IC depend [167]. Typically, they are integrated into silicon as customized security blocks that handle chip and device identities, manage cryptographic keys and functions, secure boot processes, attestation, authentication, firmware updates, etc. The hardware root of trust is expected to be capable of detecting an intrusion, disabling access pending further actions, and/or obfuscating logic operations of the IC. What lays the foundations for a silicon-based fixed-function root of trust is its authentication protocol. As an initial part of the actual challenge-response procedure, an IC creates a truly random token, commonly known as a challenge or a nonce, and sends it over to a security processor that computes a hash of the nonce. This hash (or digest) is subsequently returned to the IC to be compared with a hash value produced internally (by the IC). The latter is usually done by a device implementing a lightweight cryptographic hash function. Once the user is granted access to the test logic, SCs are used to yield encrypted or decrypted streams of test stimuli and test responses.

The previous chapters have recalled lightweight cryptographic hash functions, general purpose SCs as well as solutions designed to secure on-chip DFT ecosystems. However, the complexity of these solutions may still be considered unacceptable by many IC vendors who often face the dilemma of using an SHA engine [72] or other IP security cores where computations do not lend themselves to lightweight hardware implementations [37], [45]. Furthermore, SoC integration flow, distinguished by the prevalence of design reuse, may generate extra iterations as pre-designed intellectual property (IP) security soft cores often need to be fine-tuned and incrementally optimized until the register-transfer-level (RTL) synthesis process reaches the best trade-off between performance, area footprint, security, power, and also testability.

A hardware RoT that maintains both data integrity and hardware quality ensured by test should satisfy at least the following requirements:

- a high degree of defense-in-depth against scan attacks,
- the ability to scale by adopting cryptographic primitives of a wide range of sizes,
- full compatibility with a design and DFT flow,
- a low area overhead,
- no performance degradation of at-speed test applications.

To address the above concerns while fulfilling requirements of IC testing, lightweight, yet effective, suitable for implementation in an all-digital, standard-cell-synthesis flow, new security primitives that can be used to design a hardware RoT are proposed and analyzed in the next chapters, as summarized below.

Chapter 6 introduces hybrid ring generators (HRG) – a new class of lightweight linear finite state machines. While they are structurally similar to conventional ring generators, the new devices can circulate test data faster. This improves the performance of linear circuits used in test and security realms. Several applications of HRG such as MISRs or programable PRPGs are also discussed along with data providing architectural details of HRGs for sizes up to 1184 bits.

HRGs, working in tandem with a nonlinear sequential circuitry whose feedback network employs Boolean functions that are based on bent functions, are used to build a scalable, lightweight cryptographic family of hash functions $H_2B$ presented in Chapter 7. Two groups of tests, including the NIST test suite, confirm that the scheme can fulfill requirements for a trustworthy and cryptographically secure hash function. Furthermore, resilience against brute-force, cryptanalytic, and side-channel attacks, as well as the self-testing capabilities of the presented design is described.

Chapter 8 presents lightweight SCs that can work as standalone units or be destined for the root of trust applications. High-speed HRGs and NLFSRs work synergistically to yield output keystreams. Desired features of the ciphers were comprehensively examined using several statistical tests. It is also shown that the proposed SCs can resist various types of cryptographic attacks.

The last presented item is a lightweight true random number generator producing a nonce, as shown in Chapter 9. The scheme performance was studied with the help of hardware and simulation platforms. The randomness of the raw binary sequences without any postprocessing was tested with NIST and AIS-31 test suites. The performance of the new scheme is compared with six state-of-the-art solutions.

Finally, a hardware root of trust (RoT) is presented in Chapter 10. It works with just a few blocks whose architectural details are discussed earlier in Chapters 6, 7, 8, and 9. It easily integrates with SSN technology by taking advantage of its inherent data scrambling and packetized test data distribution. In addition to SSN-based designs, the proposed RoT can improve security of other test interfaces that employ a challenge-response authentication protocol.

# 6. Hybrid ring generators

Test compression introduced ring generators – high speed devices formed by transforming the structure of conventional LFSRs while preserving their transition functions. Ring generators feature a reduced number of levels of XOR logic, minimized internal fan-outs, and simplified layout and routing. This chapter presents *hybrid ring generators* (HRGs) [140] which take linear finite state machines to the next evolutionary level in the development of their ecosystems. While using the principal design rules of ring generators, the new devices are structurally improved with enhanced overall performance.

## 6.1 Ring generators

On-chip test data decompressors are the very first devices that had deployed *ring generators* [120], [121], [125], [146], [147] – high performance LFSRs – that quickly started carving out a reputation for themselves as versatile solutions capable of outperforming traditional schemes through an unmatched speed of operations and layout-friendly structures. Given a characteristic (feedback) polynomial, ring generators feature smaller internal fan-outs, shorter propagation paths, and simpler circuit layout and routing than popular and commonly used Fibonacci or Galois LFSRs [71], [99] whose long irregular feedback paths may limit the operating speed, cause severe frequency degradation, and may take up a considerable silicon area, especially for polynomials with a large number of terms. Fig. 6.1 recalls a basic architecture of a 32-bit ring generator with a primitive polynomial $h(x) = x^{32} + x^{28} + x^{23} + x^{20} + x^{17} + x^{12} + x^{8} + x^{4} + 1$, which causes this ring generator to go through all possible $2^{32} - 1$ nonzero values before entering a seed state. Typically, its structure can be created by forming a ring counter, and then by adding feedback taps which correspond to successive terms of a characteristic polynomial. A feedback loop associated with tap $x^k$ is made up from $k$ adjacent flip-flops, beginning with the leftmost ones (see Fig. 6.1). Note that two feedback nets cannot cross each other [121]. If one uses an appropriate characteristic polynomial, then a ring generator may assume a regular ladder-like shape. An extensive collection of such primitive polynomials is available in [144]. Since a subset of $k$ adjacent flip-flops can be chosen in different ways as long as the resultant feedback line does not cross any other feedback line, the ring generators offer an appreciable degree of flexibility in forming their structures.

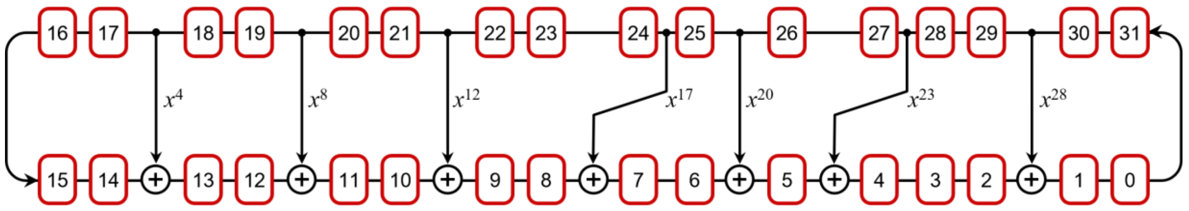


Figure 6.1 Ring generator implementing a primitive polynomial $h(x) = x^{32} + x^{28} + x^{23} + x^{20} + x^{17} + x^{12} + x^{8} + x^{4} + 1$.

In addition to nanometer test, ring generators can perform more quickly and reliably than their conventional predecessors on a wide range of problems in such areas of engineering as communications, digital broadcasting, data transmission, mobile telephony, security and cryptography, white noise generation, error detection and correction, data compression, or event counting [124]. In this thesis, new schemes have been proposed to protect scan-based designs against unauthorized usage of their test logic. These hardware roots of trust may use ring generators (or their hybrid versions) to hash proprietary data (Chapter 7), encrypt/decrypt test data streams (Chapter 8), or produce true random numbers (Chapter 9).

Ring generators re-emerged as a research topic in 2011 [188] with the observation that conventional ring generators can be rearchitected in such a way that its XOR gate count is virtually halved provided a characteristic polynomial meets certain criteria. It has given rise to a new solution termed hybrid ring generators, similarly to hybrid LFSRs of [186].

## 6.2 Hybrid linear feedback shift registers

Hybrid LFSRs reduce the number of XOR gates by combining both external-XOR and internal-XOR logic within the same register. It was shown [186] that if a characteristic polynomial can be rewritten as

$$h(x) = x^k\, b(x) + b(x) + 1, \tag{6.1}$$

where $x^k\, b(x)$ and $b(x)$ have no terms in common but $b(x)$, then a hybrid top-bottom LFSR can be set up using the following feedback:

$$F(x) = x^k\, b(x) - x^k + 1. \tag{6.2}$$

Note that symbol "–" indicates a top-tap connection back to the first stage, whereas symbol "+" indicates a bottom-tap connection to the next stage. Similarly, if a characteristic polynomial can be rewritten as

$$h(x) = x^n + x^k\, b(x) + b(x), \tag{6.3}$$

then a hybrid bottom-top LFSR can be constructed using the following feedback:

$$F(x) = x^n - x^{n-k} + b(x). \tag{6.4}$$

Again, symbols "–" and "+" are used to indicate respective tap connections. Touba and Wang have proved that if a given LFSR can be converted into a hybrid one, then the same can be done with the corresponding ring generator [187], [188]. This is illustrated in Fig. 6.2 for the 8-bit ring generator implementing a primitive polynomial

$$h(x) = x^8 + x^6 + x^5 + x^3 + 1, \tag{6.5}$$

which can be rewritten as

$$h(x) = x^3(x^5 + x^3) + (x^5 + x^3) + 1. \tag{6.6}$$

The above formula indicates that a HRG can be constructed based on the following feedback:

$$F(x) = x^3(x^5 + x^3) - x^3 + 1 = x^8 + x^6 - x^3 + 1. \tag{6.7}$$

In fact, the same result can be obtained by applying transformations moving feedback connections around LFSRs with the possibility of adding or cancelling certain XOR gates [121] should a source tap cross a destination tap (XOR gate) or vice versa. Fig. 6.2 illustrates successive steps of such a transformation. The grey arrows indicate a tap and a direction it is moved to rearrange the LFSR while preserving its characteristic polynomial. For example, the first step rotates feedback tap 6 → 2 counter-clockwise by three flip-flops. Then, feedback tap 7 → 1 is moved by two flip-flops. After step 4 two feedback taps cancel each other leading finally to an HRG with just two feedback taps as anticipated by formula (6.7). It is also worth noting that a hybrid ring defined by its feedback function $F(x)$ can be easily arranged in the same way as it is done for conventional ring generators, i.e., by encompassing a given number $k$ of flip-flops to form a given feedback loop corresponding to coefficient $x^k$.

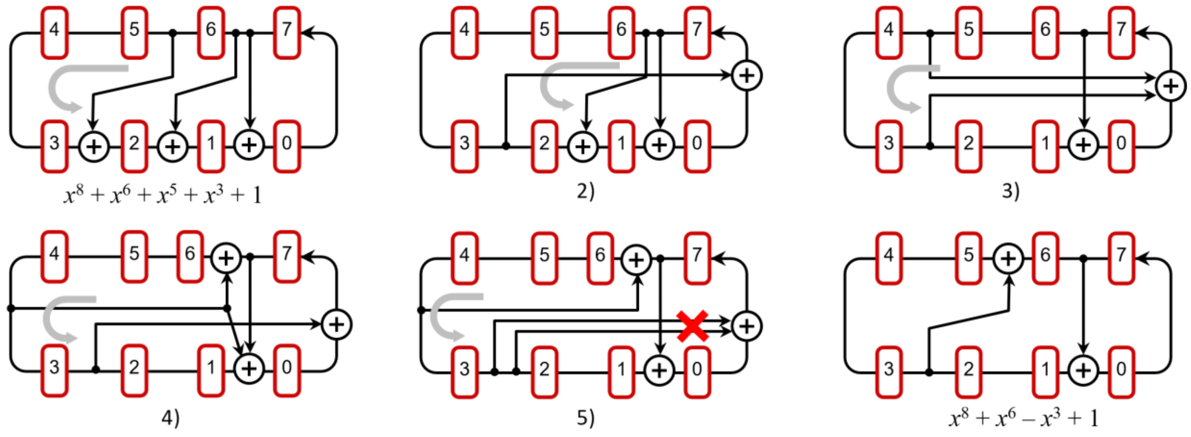


Figure 6.2 Ring generator and its hybrid version (after transformations).

By virtue of the above methods, the number $g$ of 2-input XOR gates employed by a conventional ring generator (similarly to conventional LFSRs) can be reduced to $(g + 1)/2$ provided a suitable feedback polynomial is used. In particular, an HRG with a primitive pentanomial will feature two 2-input XOR gates instead of three ones. The reference [188] offers a list of primitive pentanomials of degree up to 800 that can be employed to get HRGs having two 2-input XOR gates. No primitive polynomials, however, have been reported with more than 5 terms that meet the requirements (6.1) or (6.3). Moreover, HRGs with more aggressively reduced XOR gate counts have not been thoroughly examined yet, including those with a varying number of top-bottom and bottom-top feedback nets. The results presented in the next chapters are the first steps in this direction.

## 6.3 Basic design scheme

The HRGs of [187] have an intrinsic component: a single feedback connection going in the opposite direction than all the remaining feedback wires – compare Fig. 6.3 showing a 24-bit HRG implementing the primitive polynomial

$$h(x) = x^{24} + x^{22} + x^{19} + x^{14} + x^{12} + x^{10} + x^{7} + x^{2} + 1$$

that satisfies (6.1) since

$$h(x) = x^{12} (x^{12} + x^{10} + x^{7} + x^{2}) + x^{12} + x^{10} + x^{7} + x^{2} + 1.$$

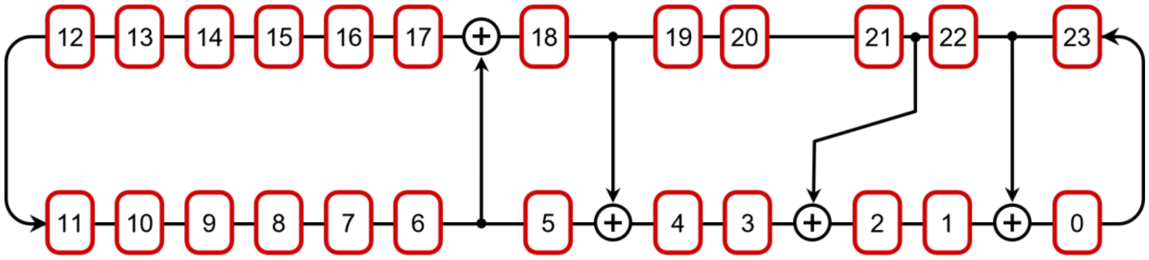


Figure 6.3 Hybrid ring generator with $h(x) = x^{24} + x^{22} + x^{19} + x^{14} + x^{12} + x^{10} + x^{7} + x^{2} + 1$ obtained by using a method of [187].

As a result, the only advantage of that hybrid ring is its reduced XOR gate count from 7 to 4, as otherwise it offers similar performance to that of the conventional ring generators. It appears, however, that the maximum length HRGs may have much more diversified layouts and may offer more substantial area savings. Consider a 32-bit maximum length HRG shown in Fig. 6.4a with the following feedback function:

$$F(x) = x^{32} - x^{28} + x^{24} - x^{18} + x^{12} - x^{5} +1.$$

It was found in $O(n^2)$ time, where $n$ is the HRG size, by using the fast LFSR simulation technique of [125]. As can be seen, this HRG features five feedback connections selected in such a way that they run alternately up and down to form two groups of oppositely disposed nets whose mutual spatial separations are roughly the same making the feedback lines

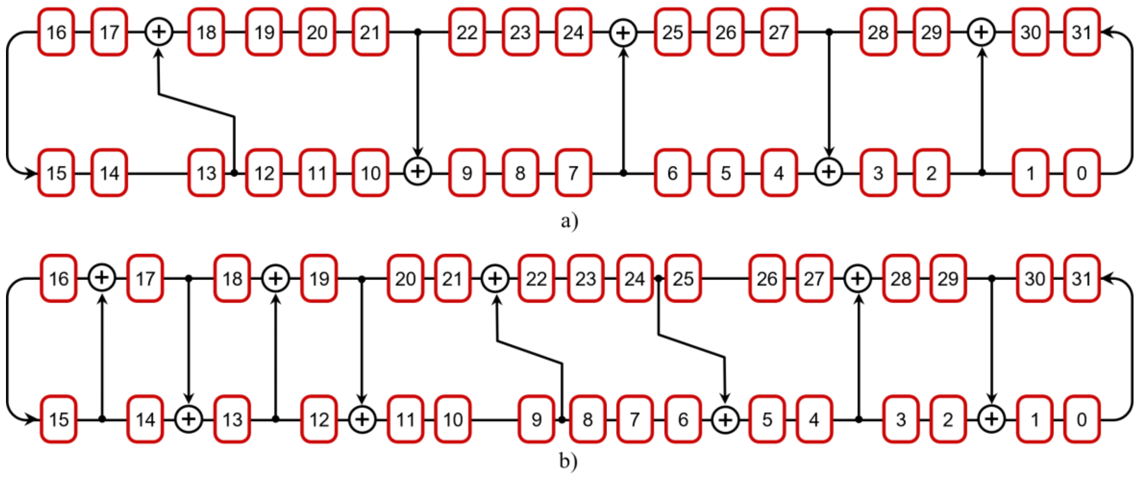


Figure 6.4 Primitive hybrid ring generators with
a) $h(x) = x^{32} + x^{28} + x^{26} + x^{25} + x^{22} + x^{21} + x^{20} + x^{19} + x^{18} + x^{16} + x^{15} + x^{14} + x^{13} + x^{11} +x^{10} + x^{9} + x^{8} + x^{6} + x^{5} + x^{4} + 1$,
and b) $h(x) = x^{32} + x^{26} + x^{24} + x^{23} + x^{21} + x^{19} + x^{18} + x^{17} + x^{16} + x^{13} + x^{12} + x^{10} + x^{6} + x^{4} + 1$.

(approximately) uniformly distributed. Interestingly, the HRG of Fig. 6.4a implements the following primitive polynomial:

$$h(x) = x^{32} + x^{28} + x^{26} + x^{25} + x^{22} + x^{21} + x^{20} + x^{19} + x^{18} + x^{16} + x^{15} + x^{14} + x^{13} + x^{11} + x^{10} + x^{9} + x^{8} + x^{6} + x^{5} + x^{4} + 1.$$

This result can be easily verified by taking an output sequence of $2n$ bits observed on any stage of the register, and then by running the Berlekamp-Massey algorithm [110] to find the minimal polynomial of that linearly recurrent sequence. In contrast to a conventional ring generator that would need 19 XOR gates (each having two inputs), the HRG of Fig. 6.4a employs just five such gates, thus achieving a 19/5 = 3.8 times reduction of the XOR gate count. Another example is illustrated in Fig. 6.4b with the enlarged number of feedback taps. This HRG uses the feedback function

$$F(x) = x^{32} + x^{28} - x^{24} + x^{19} - x^{13} + x^{8} - x^{6} + x^{4} - x^{2} + 1.$$

This is equivalent to the following primitive polynomial:

$$h(x) = x^{32} + x^{26} + x^{24} + x^{23} + x^{21} + x^{19} + x^{18} + x^{17} + x^{16} + x^{13} + x^{12} + x^{10} + x^{6} + x^{4} + 1.$$

Given the high computational efficacy of the primitiveness test [125], one can select a suitable HRG by means of the following procedure. Every iteration it works with a candidate HRG topology rather than a candidate feedback polynomial. It allows designers to pick the most preferable structure characterized by its size, a desired number of feedback taps, their orientation (up or down), their mutual separation (minimal or maximal), and other constraints that can be easily added. Given an $n$-bit HRG, the appropriate test is used to see if the examined structure can yield an $m$-sequence. Recall that this task can be accomplished in $O(n^2)$ time. If the test fails, we pick another HRG by relaxing some of the constraints, primarily locations of one or more feedback taps. Having found a maximum-length HRG, one can retrieve the corresponding characteristic polynomial by virtue of the Berlekamp-Massey algorithm, as shown earlier. It is worth noting that seeking an appropriate primitive polynomial to set up the corresponding HRG would take CPU time proportional to $O(n^2)$ as well. Unfortunately, there are no known criteria under which one could decide if a given primitive polynomial can be used to form a desired HRG. Back to Fig. 6.4, it remains to devise how to arrive with the HRGs of this figure based only on the primitive polynomials (listed in the caption) in an algebraic manner similar to that of [186].

Using this structural approach, HRGs have been identified for all degrees up to 1184. Selected results are gathered in Table 6.1. All HRGs listed there are optimal in the sense of having feedback connections distributed as uniformly as possible. They are, therefore, amenable to be highly modular structures. Furthermore, feedback taps have been selected such that they alternately go up and down to accelerate internal circulation of data (see Chapter 6.5). Note that only the exponents of nonzero terms are represented, and terms corresponding to feedback taps “going up” are underlined.

Table 6.1 Primitive hybrid ring generators, $n \leq 1184$

| | | |
|---|---|---|
| 8 5 2 0 | 280 249 217 185 154 123 92 61 30 0 | 728 648 566 484 402 321 240 158 79 0 |
| 13 8 3 0 | 288 256 224 191 158 126 94 62 31 0 | 736 654 573 490 408 325 244 161 80 0 |
| 16 14 11 7 4 0 | 296 263 230 197 163 130 96 62 31 0 | 744 661 578 495 411 329 246 162 81 0 |
| 17 13 9 6 2 0 | 304 270 235 202 167 134 99 64 32 0 | 752 667 583 498 413 328 246 164 82 0 |
| 19 17 12 8 4 0 | 312 278 242 207 172 136 102 66 33 0 | 760 676 591 506 421 337 252 167 82 0 |
| 24 19 13 8 3 0 | 320 285 249 213 177 141 107 72 36 0 | 768 671 572 472 372 276 207 138 69 0 |
| 31 25 18 13 6 0 | 328 292 255 218 183 146 110 73 36 0 | 776 690 604 517 431 345 258 171 84 0 |
| 32 26 20 13 7 0 | 336 299 264 227 189 152 114 76 38 0 | 784 699 611 523 435 347 259 172 85 0 |
| 36 31 26 19 12 7 0 | 344 307 268 231 193 155 116 77 38 0 | 792 704 617 528 439 351 263 175 87 0 |
| 40 32 23 17 9 0 | 352 313 275 235 195 156 117 77 38 0 | 800 712 623 533 444 354 266 177 87 0 |
| 44 37 28 18 10 0 | 360 321 280 239 198 157 116 78 38 0 | 808 719 629 538 447 358 268 177 88 0 |
| 48 41 32 23 14 7 0 | 368 328 286 244 202 162 120 78 39 0 | 816 725 634 543 452 361 270 180 89 0 |
| 52 43 35 27 17 7 0 | 376 335 293 252 209 166 123 81 40 0 | 824 732 641 548 457 365 272 180 90 0 |
| 56 47 37 30 20 10 0 | 384 341 298 254 211 167 124 82 41 0 | 832 727 619 511 407 304 228 152 76 0 |
| 60 51 42 33 22 11 0 | 392 349 304 261 217 173 128 85 42 0 | 840 748 654 562 469 375 281 187 93 0 |
| 61 52 42 31 22 11 0 | 400 356 311 268 223 179 134 89 44 0 | 848 755 661 566 471 376 281 186 93 0 |
| 64 56 49 40 32 23 15 7 0 | 408 351 293 233 172 116 87 58 29 0 | 856 761 666 572 476 380 285 189 93 0 |
| 68 61 52 44 35 27 18 9 0 | 416 371 325 278 231 184 137 90 45 0 | 864 769 672 575 480 383 287 191 95 0 |
| 72 64 55 45 37 28 18 8 0 | 424 377 331 284 236 188 141 94 46 0 | 872 776 678 580 484 387 289 191 95 0 |
| 76 67 57 47 37 28 18 9 0 | 432 386 337 289 240 191 142 94 46 0 | 880 784 685 586 488 389 290 192 96 0 |
| 80 70 59 50 39 29 18 9 0 | 440 392 342 292 244 194 145 95 47 0 | 888 789 691 592 494 394 294 196 97 0 |
| 84 73 63 52 41 30 19 9 0 | 448 398 347 296 246 195 145 96 48 0 | 896 796 698 599 498 398 299 199 99 0 |
| 88 78 66 54 43 32 20 10 0 | 456 406 354 302 250 198 147 98 49 0 | 904 806 705 604 503 402 301 202 101 0 |
| 89 79 68 58 47 35 24 12 0 | 464 413 360 307 254 203 152 100 50 0 | 912 796 679 562 446 340 255 170 85 0 |
| 92 80 69 57 45 33 21 10 0 | 472 420 367 314 263 210 157 105 53 0 | 920 819 716 615 513 411 308 205 102 0 |
| 96 87 76 65 55 44 33 22 11 0 | 480 420 359 300 241 180 119 58 0 | 928 825 722 618 515 411 308 205 101 0 |
| 100 90 78 66 56 45 33 21 10 0 | 488 434 379 326 271 217 163 108 54 0 | 936 832 728 623 518 413 309 206 102 0 |
| 104 92 81 69 57 45 33 21 10 0 | 496 442 388 332 276 220 166 111 55 0 | 944 840 735 629 523 417 313 207 103 0 |
| 107 96 84 72 59 46 34 21 10 0 | 504 449 393 338 282 225 168 112 55 0 | 952 846 739 633 527 421 316 209 104 0 |
| 108 96 85 72 60 47 35 23 11 0 | 512 455 397 339 281 225 169 111 55 0 | 960 853 745 638 532 424 316 210 105 0 |
| 112 102 91 80 70 60 49 38 28 18 9 0 | 520 462 406 348 289 230 171 112 56 0 | 968 860 752 645 537 429 320 212 106 0 |
| 116 107 97 87 76 66 55 44 33 22 11 0 | 521 464 405 346 288 230 171 113 56 0 | 976 857 734 612 489 368 276 184 92 0 |
| 120 109 97 85 74 62 50 38 27 18 9 0 | 528 470 411 351 291 233 173 114 57 0 | 984 860 737 619 495 372 279 186 93 0 |
| 124 113 101 91 80 68 56 44 32 20 10 0 | 536 477 416 356 295 234 174 116 58 0 | 992 882 771 663 552 441 330 219 109 0 |
| 127 116 104 93 81 69 57 45 33 22 11 0 | 544 472 398 323 250 176 132 88 44 0 | 1000 874 747 623 496 380 285 190 95 0 |
| 128 115 100 85 71 57 42 28 13 0 | 552 476 399 322 250 180 135 90 45 0 | 1008 897 785 672 560 448 335 222 110 0 |
| 132 118 102 86 71 56 40 26 13 0 | 560 498 435 373 310 247 185 122 61 0 | 1016 904 790 676 563 450 337 224 111 0 |
| 136 121 106 90 75 59 44 28 14 0 | 568 506 442 380 316 253 189 125 62 0 | 1024 910 796 682 567 453 340 226 112 0 |
| 140 125 109 93 76 60 44 28 14 0 | 576 513 448 386 323 258 194 129 64 0 | 1032 917 803 687 571 455 340 226 113 0 |
| 144 128 112 96 79 63 46 30 15 0 | 584 519 453 387 322 257 191 126 63 0 | 1040 926 809 692 577 461 345 228 114 0 |
| 148 132 115 100 84 67 50 33 16 0 | 592 525 458 391 325 258 192 128 64 0 | 1048 917 786 662 531 400 300 200 100 0 |
| 152 135 117 99 82 64 47 30 15 0 | 600 533 465 398 330 264 197 130 65 0 | 1056 939 823 705 588 470 352 234 116 0 |
| 156 140 122 104 87 70 52 34 16 0 | 607 540 472 404 336 270 203 135 68 0 | 1064 948 829 711 593 474 357 238 119 0 |
| 160 142 123 104 87 68 51 32 16 0 | 608 540 471 402 334 267 199 132 66 0 | 1072 953 833 713 594 474 355 236 118 0 |
| 168 149 131 111 92 72 53 34 17 0 | 616 533 451 370 286 208 156 104 52 0 | 1080 960 841 721 600 479 358 238 118 0 |
| 176 157 138 119 100 80 60 40 20 0 | 624 555 485 415 347 277 207 138 69 0 | 1088 968 846 724 603 483 361 239 119 0 |
| 184 164 143 123 103 82 62 41 20 0 | 632 562 491 422 351 281 210 140 69 0 | 1096 975 852 730 607 485 363 242 120 0 |
| 192 171 150 128 107 85 64 43 22 0 | 640 569 499 427 356 285 213 142 70 0 | 1104 981 858 735 612 488 366 242 121 0 |
| 200 178 155 132 109 87 65 42 21 0 | 648 576 504 431 359 286 213 142 71 0 | 1112 988 863 740 617 493 369 245 122 0 |
| 208 186 162 140 117 94 71 47 24 0 | 656 585 512 438 364 290 216 142 71 0 | 1120 982 847 708 570 432 324 216 108 0 |
| 216 193 170 145 120 95 72 48 24 0 | 664 590 515 441 367 294 219 144 72 0 | 1128 1004 878 752 629 503 378 252 126 0 |
| 224 199 173 147 122 98 72 48 23 0 | 672 598 524 448 372 297 222 147 73 0 | 1136 1010 886 759 633 506 379 254 127 0 |
| 232 206 180 154 128 101 74 48 24 0 | 680 605 528 451 375 299 224 149 74 0 | 1144 1017 891 764 636 509 382 254 126 0 |
| 240 214 187 160 134 107 80 53 27 0 | 688 598 506 416 325 240 180 120 60 0 | 1152 1024 895 766 637 508 382 255 127 0 |
| 248 221 194 167 139 111 83 55 27 0 | 696 605 514 421 329 244 183 122 61 0 | 1160 1031 901 774 644 516 386 256 127 0 |
| 256 228 199 170 142 113 85 57 29 0 | 704 627 550 471 392 313 234 155 76 0 | 1168 1038 908 777 646 517 386 257 128 0 |
| 264 235 205 175 146 118 88 58 30 0 | 712 633 554 475 395 315 235 155 77 0 | 1176 1047 917 785 653 522 390 258 129 0 |
| 272 242 212 182 151 121 91 60 29 0 | 720 640 562 481 400 319 238 159 79 0 | 1184 1053 921 790 657 524 392 261 130 0 |

For example, 32 $\underline{28}$ 24 $\underline{18}$ 12 $\underline{5}$ 0 stands for the feedback function F($x$) = $x^{32} - x^{28} + x^{24} - x^{18} + x^{12} - x^{5} + 1$ of HRG in Fig. 6.4a. Note that in addition to sizes which are multiplicities of 4 and 8, the table also includes primitive HRGs with a period equal to a Mersenne prime number, i.e., for $n$ = 13, 17, 19, 31, 61, 89, 107, 127, 521, and 607. Dividing the number of polynomial terms by the number of the corresponding HRG feedback function terms (with the exception of terms $n$ and 0) gives the XOR count reduction. It appears that this reduction can be strikingly as high as 7.57x, for $n \leq 1184$. This is for HRG

$$F(x) = x^{168} - x^{148} + x^{128} - x^{106} + x^{82} - x^{59} + x^{36} - x^{18} + 1,$$

whose feedback polynomial is

168 150 148 145 144 130 128 127 126 125 124 121 110 109 107 106 105 104 103 102 101 98 89 87 86 85 84 83 82 81 80 79 78 69 65 64 63 62 61 60 59 58 56 45 43 42 41 40 38 36 23 22 20 18 0.

The approach presented in this chapter can be easily used to obtain HRGs of many other architectures that may be required, for example, due to space, layout, or routing constraints.

## 6.4 Reciprocal and dual HRGs

This chapter briefly discusses two important aspects of deploying HRGs in applications where one needs to produce a pseudorandom sequence or its derivatives. In many instances, instead of the original pseudorandom sequence, it is necessary to employ a sequence which is exactly the reverse of the original vector. Typically, this is achieved by using LFSRs or ring generators implementing a reciprocal polynomial $h^*(x)$ of a given polynomial $h(x)$, where $h^*(x) = x^n h(1/x)$. As could be expected, given an $n$-bit HRG, one can obtain its reciprocal by converting the feedback function of the HRG the same way it is done for conventional rings. Note that all feedback connections will maintain their original directions. Consider, for example, a 32-bit maximum-length HRG shown in Fig. 6.5a. Its feedback function is given by

$$F(x) = x^{32} + x^{31} - x^{30} + x^{27} - x^{22} + x^{17} - x^{12} + x^{11} - x^{7} + x^{3} + 1,$$

where the corresponding primitive polynomial is

$$h(x) = x^{32} + x^{31} + x^{30} + x^{29} + x^{26} + x^{25} + x^{24} + x^{23} + x^{20} + x^{18} + x^{17} + x^{14} + x^{13} + x^{8} + x^{6} + x^{5} + x^{3} + x + 1.$$

If another HRG is constructed with

$$F^*(x) = x^{32} + x^{29} - x^{25} + x^{21} - x^{20} + x^{15} - x^{10} + x^{5} - x^{2} + x^{1} + 1$$

as its feedback function (Fig. 6.5b), it will generate a sequence which is the exact reversal of the sequence produced by the circuit of Fig. 6.5a. A key point here is that a reciprocal of a given HRG is obtained as its exact mirror image (compare Fig. 6.5a and 6.5b). Consider the following two feedback taps in Fig. 6.5a: $–x^{12}$ and $x^{11}$. Their reciprocals are $–x^{20}$ and $x^{21}$. Tap $x^{21}$, however, is not driven directly by flip-flop 26 (as it might be implied by a formula used to compute a reciprocal). Instead, its stem is fed by an XOR gate placed on the output of flip-flop 26 due to tap $–x^{20}$. This arrangement preserves the HRG mirror image and assures correctness of the reciprocal form. The same phenomenon occurs for taps $–x^{30}$ and $x^{31}$, and their reciprocals $–x^2$ and $x^{1.}$

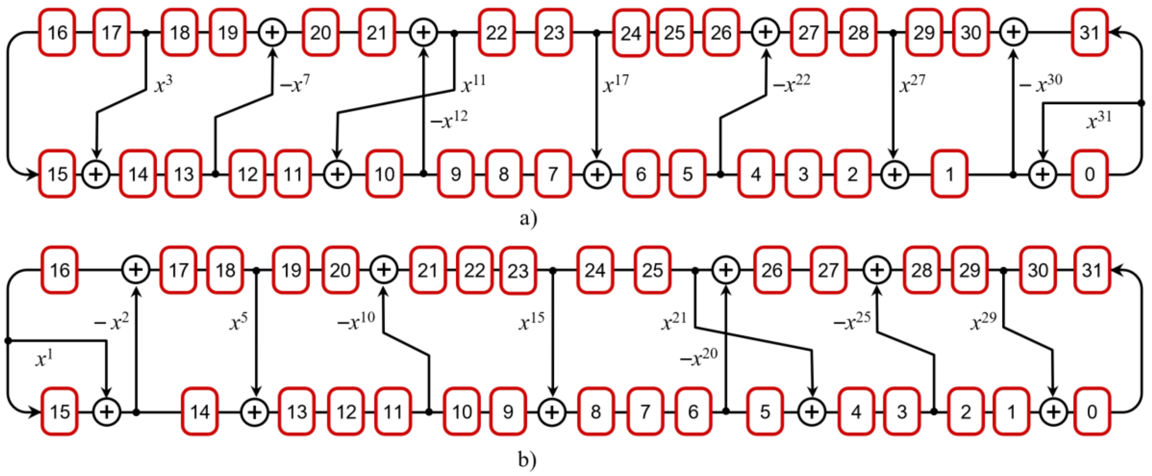


Figure 6.5 32-bit HRG (a) and its reciprocal form (b).

Similarly to reciprocal structures, every HRG has its own *dual* counterpart. Given a conventional ring generator, its dual form has the direction of all feedback connections reversed. Hence, a dual ring generator features XOR gates placed on the outputs of those flip-flops that have been used to drive feedback taps in the original circuit, while the feedback lines originate now at the former locations of the respective XOR gates. Dual ring generators are instrumental in the process of phase shifter synthesis, as shown in [145]. A phase shifter consists of an XOR network driven by a maximum-length LFSR, and is employed to spread apart shifted replicas of the same data in various outputs. Every output is driven by a linear combination of LFSR stages. It generates an *m*-sequence with a desired separation from other *m*-sequences by employing the "shift-and-add" property according to which a bitwise sum of any two shifts of an *m*-sequence is a shift of the same *m*-sequence. The actual phase shifter synthesis follows the steps presented in [145]. It appears that the same procedure can be used for any HRG. A phase shifter can be obtained by using a dual HRG the same way it is done for the conventional rings. For example, a dual HRG for a 32-bit HRG of the upper part of Fig. 6.6 is shown in the lower part of the same figure. Assuming an initial state of a dual HRG with a single logic 1, the state of the dual HRG after $q$ clock cycles is of interest as locations of logic 1s in this vector identify the outputs of the original HRG to be XOR-ed to produce a sequence spaced *q* shifts up a reference, i.e., a sequence originating from a stage pointed out by the logic 1 in the initial state of the dual circuit. The validity proof of this technique is analogous to that of [145]. Back to Fig. 6.6, logic simulation of the dual HRG for as many as 13,154 clock cycles reveals locations of four 1s in (blue) flip-flops 10, 11, 12, and 16. Hence, a sum of bits stored in these four flip-flops yields an *m*-sequence shifted by 13,154 steps relative to a reference *m*-sequence observed on flip-flop 0.

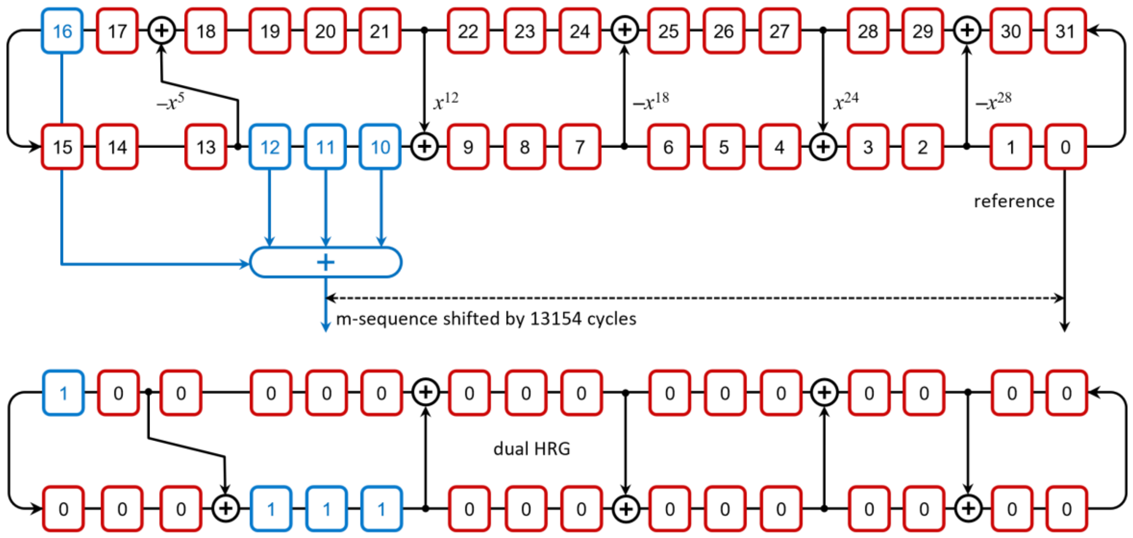


Figure 6.6 32-bit primitive HRG and its dual form used to obtain a phase shifter.

## 6.5 Multiple-input signature registers

A multiple-input signature register is one of the most straightforward applications of HRGs. MISR-based test response compaction schemes received a lot of attention in scholarly literature in the past [136]. One of the prominent results was the observation that the transient behavior of the aliasing error probability depends on architectural details of a compactor, and it can be shortened by proper selection of how injected test data circulate within a MISR. Clearly, this internal circulation can be accelerated in many cases provided an HRG is used to implement a MISR rather than a conventional ring generator. Consider a MISR of Fig. 6.7. It is built on a maximum-length 24-bit HRG with 4 input channels delivering test results through 2 injectors each. If a single error is injected through one of those channels, then one

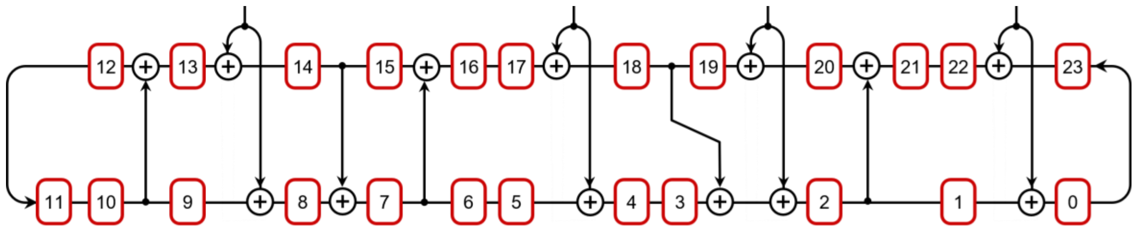


Figure 6.7 24-bit MISR driven by 4 input channels.

Table 6.2 Circulation of errors in MISRs.

| n | Input | Regular | Hybrid | Inputs | Regular | Hybrid |
|---|---|---|---|---|---|---|
| 32 | 18 | 26 | 21 | 18, 13 | 31 | 17 |
| | 20 | 26 | 23 | 20, 11 | 22 | 14 |
| | 22 | 19 | 13 | 22, 9 | 18 | 12 |
| | 24 | 19 | 15 | 24, 7 | 14 | 14 |
| 48 | 27 | 40 | 33 | 27, 20 | 47 | 26 |
| | 30 | 40 | 36 | 30, 17 | 34 | 23 |
| | 33 | 33 | 25 | 33, 14 | 28 | 20 |
| | 36 | 24 | 26 | 36, 11 | 22 | 19 |
| 64 | 34 | 63 | 40 | 34, 29 | 58 | 35 |
| | 38 | 51 | 44 | 38, 25 | 50 | 31 |
| | 42 | 51 | 48 | 42, 21 | 42 | 27 |
| | 46 | 41 | 30 | 46, 17 | 34 | 25 |
| 80 | 45 | 79 | 52 | 45, 34 | 68 | 41 |
| | 50 | 64 | 57 | 50, 29 | 58 | 36 |
| | 55 | 53 | 36 | 55, 24 | 48 | 31 |
| | 60 | 38 | 41 | 60, 19 | 53 | 59 |
| 96 | 54 | 95 | 67 | 54, 41 | 82 | 54 |
| | 60 | 78 | 73 | 60, 35 | 70 | 48 |
| | 66 | 66 | 50 | 66, 29 | 58 | 42 |
| | 72 | 46 | 55 | 72, 23 | 66 | 75 |
| 128 | 66 | 127 | 78 | 66, 61 | 122 | 73 |
| | 74 | 127 | 86 | 74, 53 | 106 | 65 |
| | 82 | 102 | 94 | 82, 45 | 90 | 57 |
| | 90 | 81 | 56 | 90, 37 | 74 | 49 |
| 160 | 90 | 159 | 105 | 90, 69 | 138 | 84 |
| | 100 | 133 | 115 | 100, 59 | 118 | 74 |
| | 110 | 103 | 69 | 110, 49 | 98 | 64 |
| | 120 | 80 | 77 | 120, 39 | 78 | 68 |
| 192 | 108 | 191 | 127 | 108, 83 | 166 | 102 |
| | 120 | 165 | 139 | 120, 71 | 142 | 90 |
| | 132 | 130 | 90 | 132, 59 | 118 | 78 |
| | 144 | 96 | 102 | 144, 47 | 94 | 79 |
| 224 | 126 | 223 | 148 | 126, 97 | 194 | 119 |
| | 140 | 185 | 162 | 140, 83 | 166 | 105 |
| | 154 | 146 | 99 | 154, 69 | 138 | 91 |
| | 168 | 146 | 113 | 168, 55 | 110 | 95 |
| 256 | 130 | 255 | 159 | 130, 125 | 250 | 154 |
| | 146 | 255 | 175 | 146, 109 | 218 | 138 |
| | 162 | 210 | 191 | 162, 93 | 186 | 122 |
| | 178 | 171 | 123 | 178, 77 | 154 | 106 |

can track its circulation within the MISR by reconstructing a part of its state trajectory beginning with a state having a single or two 1s occurring on the injection sites. A number of clock cycles $\tau$ necessary for the error to reach every flip-flop at least once can be regarded as a circulation speed metric (note that several instances of the same error may cancel each other in the course of this process). Table 6.2 provides the value of $\tau$ for several MISRs and different error injection sites (reported in the column “Input” for a single injector and in the column “Inputs” for two injectors fed by the same input channel as in Fig. 6.7). The same table contrasts HRG-based MISRs with MISRs constructed by means of regular ring generators whose architecture in each case matches feedback taps of the corresponding HRG (except their directions). The advantage of using HRG-based MISRs is clearly pronounced in each test case as they offer smaller values of $\tau$ than those of conventional rings. For example, it takes 171 cycles for an error injected into a flip-flop 178 to reach every memory element at least once in a 256-bit MISR using a regular ring generator. On the contrary, an error with the same injection pattern needs only 123 cycles to be seen at every flip-flop in an HRG-based MISR. The experiments have used the following maximum-length ring generators:

$$h(x) = x^{32} + x^{27} + x^{20} + x^{14} + x^{8} + x^{4} + 1,$$
$$h(x) = x^{48} + x^{41} + x^{34} + x^{25} + x^{16} + x^{7} + 1,$$
$$h(x) = x^{64} + x^{52} + x^{42} + x^{33} + x^{24} + x^{11} + 1,$$
$$h(x) = x^{80} + x^{65} + x^{54} + x^{39} + x^{27} + x^{13} + 1,$$
$$h(x) = x^{96} + x^{79} + x^{67} + x^{47} + x^{26} + x^{11} + 1,$$
$$h(x) = x^{128} + x^{103} + x^{82} + x^{63} + x^{40} + x^{20} + 1,$$
$$h(x) = x^{160} + x^{134} + x^{104} + x^{81} + x^{55} + x^{26} + 1,$$
$$h(x) = x^{192} + x^{166} + x^{131} + x^{97} + x^{70} + x^{37} + 1,$$
$$h(x) = x^{224} + x^{186} + x^{147} + x^{107} + x^{74} + x^{32} + 1,$$
$$h(x) = x^{256} + x^{211} + x^{172} + x^{131} + x^{84} + x^{37} + 1.$$

Note that for those conventional rings it was possible to find maximum-length HRGs with such feedback functions that their taps are identical with those of the conventional rings but directions, for example

$$h(x) = x^{32} + x^{27} + x^{20} + x^{14} + x^{8} + x^{4} + 1$$

and

$$F(x) = x^{32} - x^{27} + x^{20} - x^{14} + x^{8} - x^{4} + 1.$$

## 6.6 Programmable HRGs

Another area where HRGs improve on conventional LFSRs is in allowing simple yet effective usage of multiple characteristic polynomials. Linear devices capable of handling a number of feedback polynomials have a variety of applications that may include multiple-polynomial test data decompressors, on-chip low power test pattern generators for built-in self-test schemes, or cryptographic and security devices working with thousands of primitive polynomials. Although a conventional ring generator can be redesigned so that it allows one to pick any primitive polynomial, this solution requires many AND gates and two XOR gates interspersed between every two successive flip-flops of a lower section of a ring generator. The latter gates may slow down the entire device. Two 2-input XOR gates in a row could be replaced with a faster 3-input XOR gate, but this would be done at a price of a 30% higher transistor count [117]. Therefore, the HRG-based solution outlined in Fig. 6.8 offers a good tradeoff between the area overhead, speed, and the number of available polynomials.

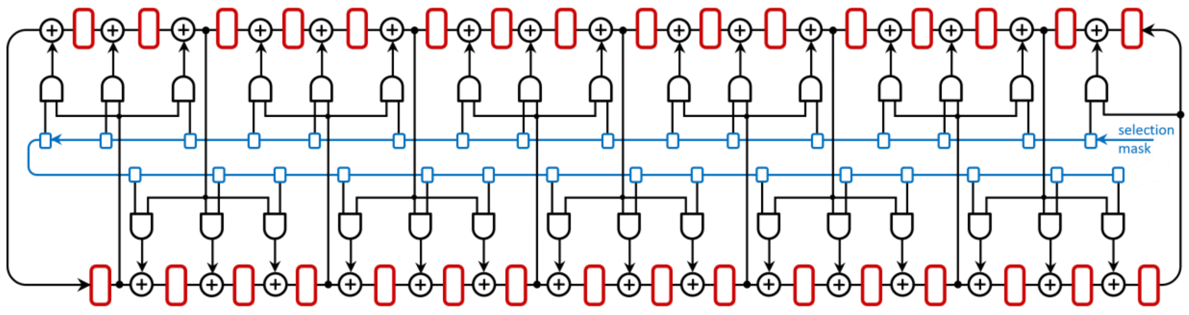


Figure 6.8 32-bit programmable hybrid ring generator.

As can be seen in Fig. 6.8, XOR logic introduces a single-gate delay (the speed of the circuit is also determined by the AND gates used to enable the actual feedback network, as in other solutions of this kind). An additional shift register (the blue flip-flops) allows one to shift-in a selection mask that determines the current feedback polynomial. Although an $n$-bit selection mask register may pick any of $2^n - 1$ feedback configurations, only some of them correspond to primitive polynomials. Table 6.3 lists the number of primitive polynomials that can be used in conjunction with HRGs similar to that of Fig. 6.8, for $n = 11, 12, \ldots, 32$. These numbers were obtained by setting all $2^n - 1$ feedback nets and running the primitiveness test. It is worth noting that different primitive HRGs may actually implement the same characteristic primitive polynomial. For such isomorphic HRGs, their common feedback polynomial is counted only once. As can be verified in Table 6.3, programmable HRGs of sizes common to many applications offer a multi-million-polynomial programming capability.

Table 6.3 Polynomial count for programmable hybrid ring generators.

| $n$ | #Polynomials | $n$ | #Polynomials | $n$ | #Polynomials |
|---|---|---|---|---|---|
| 12 | 87 | 19 | 12,776 | 26 | 648,543 |
| 13 | 346 | 20 | 11,287 | 27 | 1,585,744 |
| 14 | 433 | 21 | 39,665 | 28 | 1,789,742 |
| 15 | 1,063 | 22 | 56,015 | 29 | 3,877,201 |
| 16 | 1,166 | 23 | 95,355 | 30 | 5,311,613 |
| 17 | 2,600 | 24 | 102,608 | 31 | 20,703,016 |
| 18 | 3,466 | 25 | 487,264 | 32 | 23,881,414 |

# 7. Cryptographic hash function $H_2B$

Protection of ICs against hardware security threats has been tackled by many schemes proposed to mitigate risks associated with unauthorized access and usage of ICs in general, and intellectual property (IP) cores in particular. Typically, this is accomplished by hardware roots of trust whose crucial security primitives entail cryptographic hash functions. They provide data integrity services and thus can support the IC authentication protocols employed to counteract potential threats such as untrusted users accessing ICs. However, complexity of certain hash functions in terms of area overhead, the impact on the design flow, and testability is unacceptable. This is what motivated a new solution presented in this chapter - a simple, yet effective, lightweight, scalable cryptographic hash function $H_2B$ [139]. It employs a HRG which feeds a nonlinear sequential circuitry based on bent-like functions.

## 7.1 Basic structure

Fig. 7.1 is a block diagram of the proposed hash function $H_2B$. It consists of a HRG and a nonlinear sequential logic (NSL) receiving and processing data produced by HRG. The next paragraphs provide architectural details of these modules. It is worth noting that both parts can be initialized by means of secret keys, and a feedback function of HRG can be reprogrammed by virtue of another secret key or a secret selection mask, as explained in Section 7.1C. To a certain degree, block NSL can also be rearchitected, if required for security reasons. Given an initial (proprietary) state of HRG, $H_2B$ processes an input message of arbitrary length, e.g., a one-time nonce, in a number of clock cycles (or iterations). This number is equal to the sum of cycles needed to shift-in the input message and a predefined number of

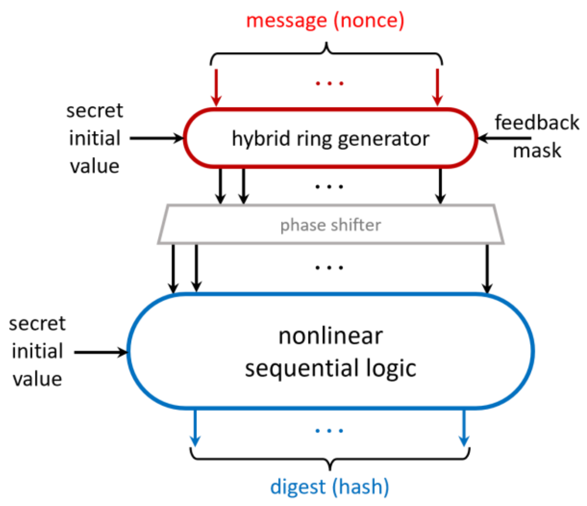


Figure 7.1 Block diagram of the proposed hash function.

additional cycles required to complete a hash computation. Hence, input bits enter HRG (in a parallel fashion through a certain number of XOR gates) where they will keep circulating until the very end. HRG-produced signals feed block NSL that works synchronously with HRG and transforms its content into a final digest. The actual hash value of a fixed length is obtained by reading out the entire content of NSL memory elements.

### *A. Hybrid ring generator*

As with conventional LFSRs, HRGs are not free from structural and linear dependencies in their output sequences. Thus, the direct use of HRG to feed the remaining $H_2B$ logic with correlated sequences may compromise the quality of results. To reduce such dependencies, a phase shifter is placed on the outputs of the generator. Also, the same circuitry acting as an expander allows one to have a relatively short HRG drive a large number of receivers. It may result in substantial savings as far as the sequential logic footprint is concerned (see also Chapter 7.2).

The phase shifter feeds a group of two-way muxes, as shown in Fig. 7.2 for a 24-bit HRG feeding a 32-bit NSL. The muxes select randomly and per-cycle data produced by HRG and its phase shifter. A signal placed on the selection input of each mux is provided by a dedicated FF of the lower ring. In addition to improved randomness, the muxes enhance nonlinearity of the results, as linear expressions leaving the phase shifter are turned into

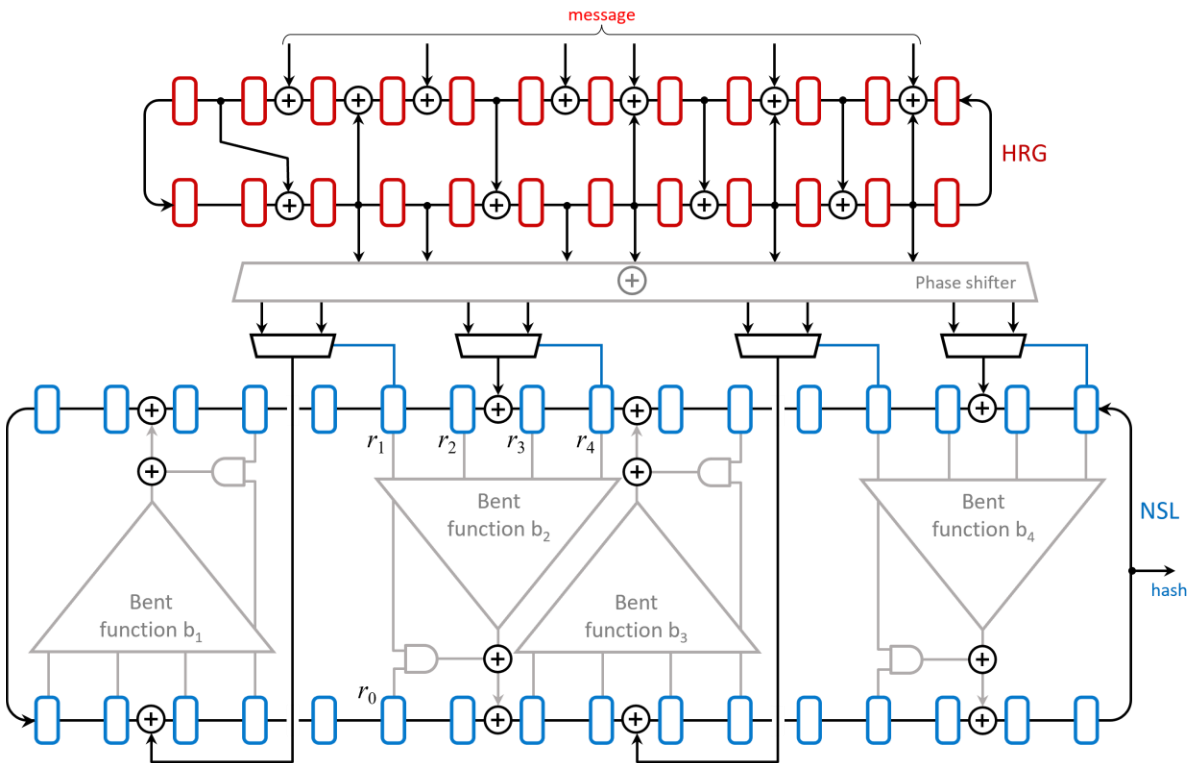


Figure 7.2 Hash function with 24-bit HRG and 32-bit NSL with four nonlinear functions.

nonlinear formulas even before they enter the lower ring and its nonlinear feedback network. What makes it possible is the equation $y = a + bc + ac$, i.e., the mux algebraic normal form (ANF) with $a$, $b$, and $c$ assuming the role of data and selection inputs, respectively.

*B. Maximal nonlinear functions*

Signals that circulate in HRG drive block NSL comprising a ring of memory elements and a feedback network which consists of different nonlinear combinational functions in 5, 7, or 9 variables. These functions are defined as follows:

$$g(r_n, \ldots, r_1, r_0) = b(r_n, \ldots, r_1) + r_k\, r_0, \quad k \in [1, \mathrm{n}], \tag{7.1}$$

where $r_i$ is an input variable provided by one of the NSL FFs, and $b(r_n, \ldots, r_1)$ is an $n$-input *bent function* [158], $n = 4$, 6, or 8. Bent functions are "the most nonlinear" among all $n$-variable Boolean functions. The degree of nonlinearity of a given function $h$ is the minimum Hamming distance between the truth tables of $h$ and an affine function. The latter one is defined as a linear function (a constant 0 or an exclusive-or of one or more variables) or its complement. In other words, the bent function is a switching function that departs from affine functions as much as possible, i.e., by $2^{n-1} - 2^{n/2-1}$. For example, $h(x_1, x_2, x_3, x_4) = x_1x_2 \oplus x_3x_4$ is a bent function which is a distance 6 from 16 affine functions of four variables (and a distance 10 from the remaining 16 affine functions).

Interestingly, the authors of [163] have shown that if a bent function $b(r_n, \ldots, r_1)$ assumes the value of zero $2^{n-1} + 2^{n/2-1}$ times, and a bent function $b(r_n, \ldots, r_1) + r_k$, $k \in [1, n]$, assumes the value of zero $2^{n-1} - 2^{n/2-1}$ times, then $g(r_n, \ldots, r_1, r_0)$ has the following properties:

1) $g(r)$ is balanced, i.e., it yields as many 0s as 1s over its input set; in other words, $g(r)$ outputs both 0s and 1s with the same probability of 0.5 provided picking any of its input vectors is equally likely,
2) $g(r)$ is highly nonlinear as its nonlinearity $N_g \geq 2^n - 2^{n/2}$; the nonlinearity of a function $g$ is the minimum number of its truth table entries that have to change to convert $g$ to an affine function, i.e., a linear function or its complement,
3) g(r) satisfies the strict avalanche criterion, i.e., any single input change causes the output change with the probability of 0.5.

These properties are considered desirable for several cryptographic primitives, including hash functions, as they can make them less vulnerable to certain algebraic attacks (see Chapter 7.3). Fig. 7.2 illustrates how four 5-input nonlinear functions based on four bent functions [158] $b_1, \ldots, b_4$ receive their $(n + 1)$ input signals. Every bent function is driven by a continuous subset of FFs of the lower ring. We also place a driver of variable $r_0$ (see above) close to a FF driven by a given nonlinear function (an example of such an assignment is shown in Fig. 7.2 for the bent function $b_2$). Moreover, every stream of data produced by the phase shifter is injected in the middle of a segment of FFs assigned to one of the bent functions. Usually, a single injector is allocated to a dedicated segment unless the number of injectors is greater than the number of segments. If so, some segments can be used more than once. Finally, all functions of NSL use its FFs in a uniform manner by having a given FF either drive just a single variable $r_1$ or receive data from just a single feedback function. Clearly, FFs fed by the phase shifter and acting as inputs to the bent functions are exceptions.

Selection of NSL functions $g(r)$ is carried out in such a way that the associated bent functions are pairwise different and they are not complements of each other. The same process is further guided by algebraic normal forms of $g(r)$. Consider, for example, all 3,584 functions compliant with (8) and having 5-inputs. Putting one of them into ANF yields the following expression:

$$g(a, b, c, d, e) = d + ae + bc + de.$$

It is comprised of 3 additions and 3 multiplications. The same group contains also a function whose ANF is as follows:

$$g(a, b, c, d, e) = a + d + ac + bc + bd + be + cd + ce + de.$$

It features 7 multiplications and 8 additions. Given the same number of inputs, it is preferable to choose functions that offer the smallest number of AND operations to enable synthesis of area-efficient hash functions. Indeed, results of logic synthesis confirm a noticeable correlation between the number of multiplications and, in a lesser extent, additions in ANF, and the resultant silicon area occupied by a given function (see also Chapter 7.2). Moreover, simulation experiments show that both the degree and the complexity of Boolean formulas in input variables (bits of a message) representing bits of a digest, i.e., NSL FFs, depend on the number of iterations rather than the number of monomials a given function $g(r)$ consists of.

## *C. Programmability*

The proposed solution can be easily turned into a keyed hash function. In addition to secret values that can be used to initialize HRG and FFs of NSL, the scheme of Fig. 7.2 can also be redesigned in such a way that HRG becomes a programmable unit capable of working with thousands of primitive feedback polynomials, as shown in Fig. 7.3. In this case, the current feedback function (and thus a feedback polynomial) is determined by a secret selection mask.

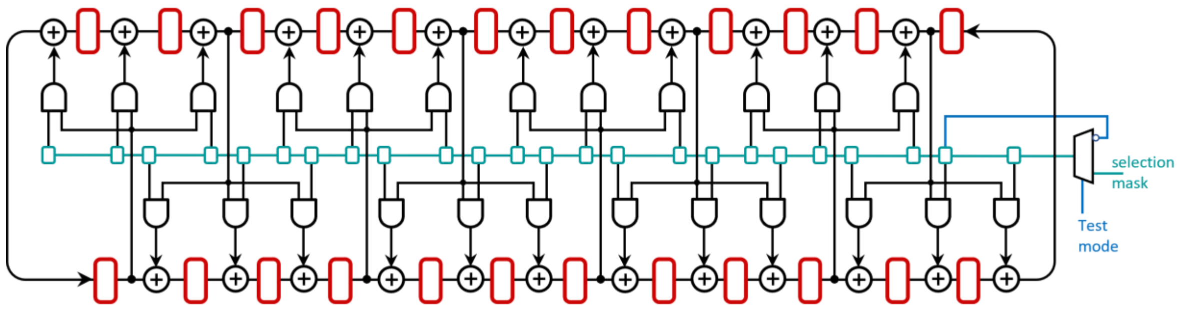


Figure 7.3 Programmable 24-bit hybrid ring generator.

This programmable HRG is a slightly modified version of the one presented in Fig. 6.8. An additional mux increases the HRG testability, employing 2-bit twisted ring counter in a test mode (see also Chpater 7.4).

## 7.2 Experimental results

The proposed hashing scheme was validated by means of statistical tests, including a test suite [12] from the National Institute of Standards and Technology (NIST). Several instances of $H_2B$ were examined by varying the size of both HRG and NSL, as well as by choosing different bent functions to implement block NSL. In order to pass all tests, it is essential for $H_2B$ to iterate for at least $c$ cycles, where $c$ is greater than the size $b$ of the NSL ring register. Following this experimental observation, the number of clock cycles needed to complete the hashing process once the entire message is uploaded to HRG was set to $2b$. Typically, N digests subjected to all tests were obtained by hashing N $s$-bit tokens (messages) produced by the Mersenne Twister pseudorandom number generator, where $s$ was chosen to be two times longer than the NSL register, e.g., a hash function with a 128-bit NSL register was fed by 256-bit messages. In particular, at least $10^9$ bits were collected on the NSL outputs, thereby producing 1 000 consecutive sequences, each comprising $10^6$ bits, to meet the requirements of the NIST tests. The same sequences were used to run the remaining tests. Hence, given a $b$-bit NSL, the actual number N of random tokens and the corresponding digests was equal to $\lceil 10^9/b \rceil$. The following sections briefly introduce each test and subsequently discuss the corresponding experimental results.

*Probability of bit values.* The simplest test is aimed at checking whether the logic value of 1 occurs on every bit position of a digest roughly half of the time (50%). If $P_1(k)$ is the probability of having 1 on bit $k$, then:

$$P_1(k) = C_k / N, \tag{7.2}$$

where $C_k$ is the 1s count on bit $k$. For large $b$, the sample mean S of $P_1(k)$, $k = 0, 1, \ldots, b - 1$, will be approximately normally distributed under the null hypothesis that the $P_1(k)$'s are independent and identically normally distributed random variables. Let $\sigma$ be the sample standard deviation. Then, the test statistic

$$Z = \sqrt{b}(S - 0.5)/\sigma \tag{7.3}$$

is normally distributed, and the test passes if $|Z| < 1.96$ given a 95% confidence level. Moreover, all 1s counts $C_k$, $k = 0, 1, \ldots, b - 1$, can also be statistically examined to see how closely they resemble a uniformly distributed random variate. This hypothesis can be verified by, for instance, the chi-square test. It works with a $b$-bin histogram of 1s observed on successive bits of N $b$-bit digests and uses the statistic

$$\chi^2 = \sum_{k=0}^{b-1} \frac{(C_k - N/2)^2}{N/2} \tag{7.4}$$

which will be approximately chi-square distributed with $b$ – 1 degrees of freedom under the null hypothesis as above.

*Buckets test*. In much the same way as bit histograms, buckets histograms show how many times hash values regarded as unsigned integers land in a particular bucket, modulo $m$, where $m$ may assume different values, e.g., $10^3$ or $10^4$. To run every instance of this test, we form an $m$-bin histogram, and then for every digest $d$ compute the corresponding value of ($d$ mod $m$) indicating a bin that is to be incremented. Finally, the chi-square test for $m$ – 1 degrees of freedom is invoked to test the histogram content for uniformity.

*Weak avalanche effect* (*diffusion and confusion test*). The avalanche effect refers to a behavior where a small change in the input results in a significant change in the output, making it statistically indistinguishable from random. A hash function demonstrates this effect if a single flipped input bit leads to approximately half of the hash bits being flipped at randomly distributed locations. A weak avalanche test proceeds as follows. Let H be an $b$-bin histogram, where $b$ is the digest size. For every random token $r$, it flips a randomly selected bit of $r$ to obtain token $r$', determines the corresponding digests $d$ and $d$', and computes D = $d \oplus d'$. Next, it increments H($k$) provided $k$-th bit of D is set to 1. As can be seen, H($k$) counts the number of times the output bit $k$ has flipped in response to a single-bit change in the input. Finally, once N tokens have been generated, we examine statistically how closely H resembles a uniform distribution. To verify this hypothesis, one can use the chi-square test with $b$ – 1 degrees of freedom and the expected value of each bin being equal to N/2, as in (7.4).

*Strong avalanche test*. This test is a generalization of the former test to all $s$ bits of the examined tokens. As a result it yields $s$ statistics. For every random token $r$, it first produces $s$ tokens $r$' by flipping all bits of $r$, one at a time. Next, the test is carried out exactly as for the weak avalanche effect. That is, for every pair ($r$, $r$') it computes a pair of digests ($d$, $d$') and determines the value of $d \oplus d'$. After repeating the experiment $\mathrm{N} \times s$ times, it uses the statistic of (7.4) to verify the hypothesis that histograms corresponding to successive input bits represent uniformly distributed random variates. Only the worst case is reported, i.e., the largest value of $\chi^2$ among all $s$ statistics collected.

*Correlation*. To validate whether a given hash function yields independent random binary combinations one can measure a correlation between any pair of bits across all N examined digests, collecting $b(b – 1)/2$ correlation coefficients. Clearly the correlation coefficient

$$\rho_{i,k} = \mathrm{N}^{-1} \sum (x_i - 0.5)(x_k - 0.5) \tag{7.5}$$

between bits $x_i$ and $x_k$ should be close to 0 to confirm that there is no strong, discernible, and systematic relation between these two positions. Such a result should hold for all pairs of bits. Due to the large number of correlation coefficients, the data regarding their mean value S over all pairs ($i$, $k$) is reported by using the test statistic

$$Z = \sqrt{b(b - 1)/2}(S - 0)/\sigma \tag{7.6}$$

to verify a hypothesis regarding the normality of S.

*Collision test*. This test begins with a randomly produced token and the corresponding digest. These steps are then repeated $n$ times until a newly produced digest matches one of the earlier generated hash values. The value of $n$ is then recorded, and the entire process repeats N times to get the mean value of steps until a collision. To pass this test, the mean number of steps before the collision should not be smaller than $2^{b/2}$, as indicated by the birthday paradox. It is worth noting that this particular test can only be run for relatively small hash functions as getting statistically significant outcomes for $b > 40$ becomes computationally infeasible.

*Architectural details.* Table 7.1 lists architectural details of hybrid ring generators and nonlinear combinational functions used to run the validation experiments reported in Chapter 7.3. A feedback function of a given $n$-bit HRG is represented as an $(n + 1)$-bit sequence with locations of 1s corresponding to respective coefficients, further encoded as a hexadecimal number. Also, a repetitive occurrence of the same digit is replaced with a decimal exponent as follows (note that a coefficient next to $x^0$ comes with a minus sign; the remaining coefficients get the alternating signs but $x^n$):

$$f(x) = x^{32} - x^{27} + x^{18} - x^{12} + x^{8} - x^{4} + 1$$

$$1\ 0000\ 1000\ 0000\ 0100\ 0001\ 0001\ 0001\ 0001_B = 108041^4{}_H$$

Nonlinear functions are represented by their truth tables, again written in a hexadecimal format. For instance, a 5-input function whose successive outputs are:

0101 0101 0101 0100 1010 1010 1010 1001

is encoded as $5^3 4A^3 9$. All information is gathered in Table 7.1, where each row corresponds to a row with the same index in Table 7.2. Due to space constraints, Table 7.1 reports architectural details for the first 21 hash functions of Table 7.2.

*Results*. The outcomes of all tests summarized above are listed in Table 7.2. Essentially, the columns of the table, except the first two, correspond to tests recalled in the column headings. The first column gives the size of HRG and NSL in terms of their memory elements. The second column lists the number of 5-input nonlinear functions used by the NSL feedback network. The column "probability of bits" reports the test statistic Z given by (7.3); this test passes if $|Z| < 1.96$. The next two columns give the observed values of the chi-square statistic of (7.4). Here, a critical test value depends on either the number $b$ of bits or the number of buckets $m$ (in the reported experiments $m = 1000$). For example, for $b = 32$ (hence the number of degrees of freedom $v = 31$) and the significance level $\alpha = 0.05$, we get $\chi^2 = 44.985$. If $m = 1000$, then $\chi^2 = 1073.643$. For the avalanche tests, Table 7.2 also reports the test statistic of (7.4). As the strong avalanche test produces $s$ different values, the table only reports the largest one. The correlation test is summarized by the metric of (7.6) which is expected to be normally distributed. Thus again the test passes provided $|Z| < 1.96$. As can be easily verified, all instances of $H_2B$ reported in Table 7.2 pass all tests. Similar results were obtained for

Table 7.1 Experimental test setups.

| ID | HRG | Nonlinear function of NSL |
|---|---|---|
| 1 | 104102081 | 1EB$^2$1E4$^2$, 6ACF65C |
| 2 | 1$^2$2$^2$1$^2$089 | AFC650C6, 9F35903A, 39FAC90A |
| 3 | 12491$^5$ | 8D7D$^2$828, B128E47D, 1BD728E4, 3A09F6C5 |
| 4 | 1$^2$0850845 | 1EB$^2$1E4$^2$, 6ACF65C |
| 5 | 1084$^2$215$^2$ | AFC650C6, 9F35903A, 39FAC90A |
| 6 | 12524924B | D$^2$2D2$^3$D, A6C0A63F, 5C936FA, 2$^3$DE$^3$1 |
| 7 | 1$^2$084215$^2$ | 5063FAC9, 8D82D8D7 |
| 8 | 128A2$^4$B | 7$^2$4B8$^2$4B, C50635F6, E1$^3$E$^2$1E |
| 9 | 148$^3$92AB | FAC90A39, 1EB$^3$41$^2$, CF9AC095, 59FCA90C |
| 10 | 152494925 | 82$^2$7D7$^2$2, 7B8B7484 |
| 11 | 124$^3$52AB | 7$^2$4B8$^2$4B, C50635F6, E1$^3$E$^2$1E |
| 12 | 1$^2$084$^3$89 | FAC90A39, 1EB$^3$41$^2$, CF9AC095, 59FCA90C |
| 13 | 12$^2$8$^2$92AB | C95FC9A0, 6ACF65C, D8EB1427, B4$^3$1E$^3$, 356A9FC, 39A06CF5, 27EB14D8, 1427D8EB, 1EB$^2$1E4$^2$,FAC905C9, D4DB242B, 935FA06C |
| 14 | 1248$^4$AB | 182BD4E7, FAC90A39, D2E$^2$1E2$^2$, 8$^3$7$^2$87$^2$, A093F5C6, F5C60536, 8$^2$2D7$^2$2D, 7$^2$D2$^3$87, A06CF539, FC9A3056, FA390539, 9ACF95C, A903A9FC, D2$^3$87$^3$, 8B1247DE, D2$^3$D$^2$2D, 218BED47, F59C0A9C |
| 15 | 18912$^3$AB | DED1$^2$21D, 1$^2$87E$^2$87, 93A0935F, 3509C5F9, 1E1$^2$D2D$^2$, 590C59F3, AC6FA36, 359FC59, B7B84748, EB4E41E4, 6FA360AC, 2$^3$D7$^3$8, 39AF3950, 90359F3A, 7B474874, EDE2$^2$12E, A39F539, 359F3A90, F9AC095C, 5F93A093, C9FA6350, D8D78D82, 8D7D$^2$828, D2E$^2$1E2$^2$ |
| 16 | 12$^4$91$^3$ | 35F906CA, DE2E12E2, 82E4D7B1, 74$^2$847$^2$B, 2714EBD8, AF63A06C, 1E4$^2$B4E$^2$, 9F35CA60, F9AC095C, 1B284E7D, 60536F5C, 4$^3$B$^3$4B |
| 17 | 14892$^3$AB | 414E$^2$BE4, 84E2B7D1, 509C5F93, 3A6F356, 6305C9AF, F6A30653, AFC950C9, ED47218B, 28$^2$D7D$^2$8, 2$^2$78D$^2$78, D2$^3$D$^2$2D, A90365CF, 717E818E, D$^2$878$^2$D2, 356FC56, 90AC9FA3, 9FA390AC, EB4E41E4 |
| 18 | 149A4A493 | 8D7D$^2$828, B128E47D, 1BD728E4, 3A09F6C5, DB718E24, 6530CF9A, 28E41BD7, DE1D12D1, D121$^2$DED, E2ED121D, C56F903A, ACF95C09, 1D$^2$ED1$^2$2, 56CFA6C, 1$^2$B4B$^2$1E, 8D41BE72, 878$^2$7$^3$8, 1$^2$2DE$^2$2D, 1DB72E84, 3A9F359, C60A935F, F3950C95, ED47B812, 4B4$^2$E1E$^2$ |
| 19 | 10$^2$8020$^2$80$^2$ 80$^2$801 | FAC905C9, 6ACF65C, D8EB1427, B4$^3$1E$^3$, 356A9FC, 39A06CF5, 27EB14D8, 1427D8EB, 1EB$^2$1E4$^2$, D4DB242B, C95FC9A0, 935FA06C, 6530CF9A, A6F3A60C, 1BD728E4, 2E841DB7, 2BE718D4, 95C0A6F3, 6AC059F3, B$^2$4B4$^3$B |
| 20 | 10201010$^2$8 0402041 | 7$^2$4B8$^2$4B, C50635F6, E1$^3$E$^2$1E, 4BE$^3$14$^2$, C9F5C90A, 95CF9AC0, 27D7$^2$282, 2D4$^2$2DB$^2$, CF9A6530, 2$^2$B4D$^2$B4, A930A9CF, 2D7$^3$82$^2$, E12$^3$DE$^2$, F5930A93, 90AC9FA3, 4ED71B82, 82E4D7B1, 428E71BD, 5C60536F, 1BEB$^2$141, 8$^2$787$^3$8, 4E827DB1, ED1DE212, B848B747, 4B$^3$4B4$^2$, FAC60A36, 1$^2$4BE$^2$4B, 41B1EB1B, 84E2B7D1, A0635F63, |
| 21 | 1020202020 4081021 | FAC90A39, 1EB$^3$41$^2$, CF9AC095, 59FCA90C, DE8B7421, 3056FC9A, B81274DE, 6C0A6CF5, 5C6F5360, BD4DB242, E248B71D, 1$^3$E$^2$1E$^2$, 5309A3F9, 2D1$^2$E1D$^2$, 17E718E8, E7E81718, D78D82D8, 7484$^2$7B7, 82728D7D, 8D827D72, FCA903A9, 95C0953F, FAC905C9, E4EB4E41, 6A0C593F, AF63A06C, 56F3A603, 6A30653F, CF95C09A, E$^3$12$^3$D, 9C50935F, E72BD418, 2E841DB7, 6F35603A, A3F95309, 50635F6C, 5F39A039, 6AF359C0, 4E7D82B1, 3A6FCA6 |

other hash functions; their detailed results are not shown here because of space constraints. Finally, the average number of steps before a collision occurs for $b = 32$ were recorded; note that this number is expected to be above the $2^{b/2} = 65\,536$ level. For the first three hash functions of Table 7.2 the results are: 82 531, 82 314, and 80 818. These numbers clearly demonstrate that the functions are collision resistant.

*NIST Test Suite* SP800-22. It consists of 15 test cases [12] that capture various types of non-randomness in a tested sequence. There are two results reported for each test. The first

Table 7.2 Test results for selected instances of $H_2B$ hash function.

| ID | Size HRG / NSL | NSL net | Probability of bits (10) | *b*-bin histogram (11) | Buckets *m* = 1000 | Weak avalanche (11) | Strong avalanche (11) | Correlation (13) | Critical values for (11) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | | 2 | 0.341408 | 15.171714 | 967.929472 | 20.857204 | 26.852667 | 0.793266 | |
| 2 | 32 / 32 | 3 | 0.348225 | 16.798651 | 1001.725632 | 20.496672 | 25.483826 | 0.249894 | 44.98534 |
| 3 | | 4 | 0.673337 | 16.17676 | 981.926464 | 20.336394 | 28.869524 | 0.383264 | |
| 4 | | 4 | 1.065126 | 42.642986 | 985.258496 | 31.624812 | 51.687925 | 1.591422 | |
| 5 | 32 / 64 | 6 | 1.723612 | 35.354599 | 960.810752 | 32.350022 | 54.756885 | 0.202635 | 82.52873 |
| 6 | | 8 | 1.3848 | 30.848388 | 951.933952 | 36.762751 | 44.227083 | 1.379746 | |
| 7 | | 6 | 1.316382 | 49.286045 | 967.252204 | 46.450458 | 73.661659 | 0.837773 | |
| 8 | 32 / 96 | 9 | 0.348326 | 45.50346 | 994.269867 | 44.894339 | 66.963025 | 0.397192 | 118.75161 |
| 9 | | 12 | 0.209959 | 43.594496 | 956.403436 | 53.352724 | 64.341161 | 0.1097 | |
| 10 | | 8 | 0.513635 | 70.351874 | 962.317312 | 60.245379 | 85.330188 | 1.402851 | |
| 11 | 32 / 128 | 12 | 0.589085 | 67.618693 | 994.60096 | 62.572724 | 85.472336 | 1.004233 | 154.30152 |
| 12 | | 16 | 1.460256 | 67.721048 | 970.684672 | 65.259611 | 88.169214 | 0.46436 | |
| 13 | | 12 | 0.058431 | 103.441275 | 928.21283 | 101.792872 | 123.886677 | 0.033146 | |
| 14 | 32 / 192 | 18 | 0.541091 | 116.042219 | 922.640607 | 114.17055 | 130.351176 | 0.593658 | 224.24462 |
| 15 | | 24 | 1.113679 | 89.409325 | 1010.542804 | 92.064664 | 130.33149 | 0.63668 | |
| 16 | | 16 | 1.842263 | 135.679799 | 1057.012096 | 140.924885 | 166.756587 | 0.405582 | |
| 17 | 32 / 256 | 24 | 0.927142 | 117.957137 | 982.09728 | 129.330056 | 160.925416 | 1.392799 | 293.24783 |
| 18 | | 32 | 1.042233 | 144.349938 | 1024.012672 | 130.653598 | 174.616201 | 0.525602 | |
| 19 | | 20 | 0.184487 | 169.817171 | 982.09856 | 162.15213 | 199.359754 | 0.121529 | |
| 20 | 64 / 320 | 30 | 0.139033 | 166.434921 | 986.43584 | 152.88024 | 212.308449 | 0.391985 | 361.65239 |
| 21 | | 40 | 0.085367 | 171.152361 | 1028.7616 | 153.663869 | 200.283871 | 1.749202 | |
| 22 | | 24 | 0.119153 | 187.555307 | 977.251886 | 194.789192 | 232.267807 | 0.771511 | |
| 23 | 64 / 384 | 36 | 0.30836 | 196.357552 | 1070.360354 | 194.943027 | 254.049255 | 0.810477 | 429.63249 |
| 24 | | 48 | 0.748154 | 189.708935 | 941.473842 | 171.172819 | 242.768323 | 0.157293 | |
| 25 | | 28 | 0.366544 | 241.595812 | 973.174457 | 226.121026 | 278.82778 | 1.092339 | |
| 26 | 64 / 448 | 42 | 0.79082 | 220.598005 | 946.939578 | 241.547596 | 279.844239 | 0.856726 | 497.29136 |
| 27 | | 56 | 0.299733 | 205.334778 | 959.680697 | 194.959089 | 275.479326 | 0.431033 | |
| 28 | | 32 | 0.957879 | 236.528548 | 996.176576 | 297.408617 | 312.834819 | 0.365718 | |
| 29 | 64 / 512 | 48 | 0.366708 | 254.377511 | 941.661888 | 282.956952 | 329.396434 | 0.526766 | 564.69613 |
| 30 | | 64 | 0.480779 | 253.374794 | 973.382336 | 247.546911 | 315.368642 | 1.677126 | |
| 31 | | 40 | 0.369358 | 332.679631 | 937.48096 | 331.727029 | 385.674723 | 1.501287 | |
| 32 | 96 / 640 | 60 | 0.293558 | 323.976118 | 988.96256 | 297.281015 | 378.35805 | 0.330312 | 698.91697 |
| 33 | | 80 | 0.156441 | 318.579505 | 960.39552 | 315.890412 | 373.951659 | 0.643606 | |
| 34 | | 48 | 0.798682 | 385.520556 | 990.396122 | 365.060894 | 452.516814 | 0.071368 | |
| 35 | 96 / 768 | 72 | 0.617124 | 424.938139 | 943.76165 | 362.100749 | 448.39882 | 0.045897 | 832.53968 |
| 36 | | 96 | 0.161761 | 383.813599 | 991.569625 | 385.940107 | 446.586431 | 0.806889 | |
| 37 | | 56 | 0.059967 | 495.341478 | 960.764911 | 446.291202 | 531.418474 | 0.536629 | |
| 38 | 128 / 896 | 84 | 0.100422 | 441.363016 | 954.734834 | 447.146135 | 523.010442 | 0.65522 | 965.70946 |
| 39 | | 112 | 0.631393 | 449.169172 | 912.28596 | 468.867435 | 519.280321 | 1.051598 | |
| 40 | | 64 | 1.474067 | 493.549278 | 977.460779 | 482.413517 | 586.933893 | 0.225016 | |
| 41 | 128 / 1024 | 96 | 0.097678 | 529.775066 | 953.794103 | 469.463117 | 597.355719 | 0.513724 | 1098.52078 |
| 42 | | 128 | 0.189449 | 511.86527 | 971.955758 | 540.722908 | 587.341028 | 0.527427 | |

one is the fraction of sequences that pass a test. Given 1 000 sequences and the significance level $\alpha = 0.01$, the pass rate is considered acceptable if it belongs to a range [0.980, 0.999]. The second result (PoP-value, i.e., a *p*-value of *p*-values) reports the uniformity of the distribution of *p*-values obtained for each individual $10^6$-bit long sequence. Hence, all 1 000 individual *p*-values are gathered in a 10-bin histogram where they are tested for uniformity. A sequence passes an individual test provided the resultant *p*-value $\geq 0.01$. The entire sequence passes a given test provided the corresponding PoP-value > 0.0001. The results of applying the NIST SP800-22 tests to binary sequences produced by four selected hash functions are listed in Table 7.3. A number in brackets that follows the test name gives the total number of subtests a given test consists of. As can be seen, again all examined hash functions pass all tests. The same applies to the remaining instances of $H_2B$ reported earlier in Table 7.2.

Table 7.3 Results for 1,000 1M-bit samples under NIST SP800-22 tests.

| | $H_2B$-128 (12) | | $H_2B$-256 (18) | | $H_2B$-512 (30) | | $H_2B$-1024 (42) | |
|---|---|---|---|---|---|---|---|---|
| Test | Pass rate | PoP-value | Pass rate | PoP-value | Pass rate | PoP-value | Pass rate | PoP-value |
| Frequency - monobit (1) | 991/1000 | 0.401199 | 996/1000 | 0.19692 | 992/1000 | 0.230755 | 995/1000 | 0.437274 |
| Block frequency (1) | 985/1000 | 0.116746 | 988/1000 | 0.504219 | 989/1000 | 0.43359 | 994/1000 | 0.737915 |
| Cumulative sums (2) | 993/1000 | 0.092041 | 997/1000 | 0.255705 | 995/1000 | 0.033584 | 995/1000 | 0.55442 |
| | 995/1000 | 0.39594 | 994/1000 | 0.331408 | 993/1000 | 0.664168 | 993/1000 | 0.544254 |
| Runs (1) | 989/1000 | 0.653773 | 994/1000 | 0.538182 | 987/1000 | 0.201189 | 988/1000 | 0.703417 |
| Longest runs (1) | 996/1000 | 0.894918 | 987/1000 | 0.995162 | 989/1000 | 0.775337 | 990/1000 | 0.41184 |
| Matrix rank (1) | 991/1000 | 0.506194 | 993/1000 | 0.731886 | 985/1000 | 0.626709 | 990/1000 | 0.512137 |
| DFT - spectral (1) | 989/1000 | 0.618385 | 992/1000 | 0.268917 | 988/1000 | 0.390721 | 984/1000 | 0.002028 |
| Non-overlapping template (148) * | 983/1000 | 0.094285 | 982/1000 | 0.55646 | 981/1000 | 0.060875 | 980/1000 | 0.293952 |
| | 996/1000 | 0.546283 | 996/1000 | 0.120207 | 996/1000 | 0.328297 | 997/1000 | 0.771469 |
| Overlapping template (1) | 992/1000 | 0.055714 | 986/1000 | 0.394195 | 993/1000 | 0.872425 | 991/1000 | 0.502247 |
| Maurer's universal (1) | 988/1000 | 0.267573 | 992/1000 | 0.100109 | 991/1000 | 0.313041 | 989/1000 | 0.004908 |
| Approx. entropy (1) | 987/1000 | 0.989055 | 993/1000 | 0.397688 | 989/1000 | 0.548314 | 985/1000 | 0.727851 |
| Random excursions (8) | 593/602 | 0.23276 | 632/641 | 0.570728 | 618/619 | 0.562457 | 613/619 | 0.572544 |
| | 596/602 | 0.366918 | 633/641 | 0.801923 | 611/619 | 0.289435 | 613/619 | 0.586055 |
| | 597/602 | 0.736578 | 634/641 | 0.61598 | 611/619 | 0.116054 | 613/619 | 0.903699 |
| | 593/602 | 0.334538 | 634/641 | 0.962407 | 612/619 | 0.569177 | 613/619 | 0.734986 |
| | 593/602 | 0.763025 | 634/641 | 0.945201 | 611/619 | 0.01265 | 614/619 | 0.955982 |
| | 596/602 | 0.122325 | 636/641 | 0.732074 | 613/619 | 0.306059 | 613/619 | 0.684788 |
| | 596/602 | 0.540878 | 632/641 | 0.943598 | 613/619 | 0.277963 | 611/619 | 0.072198 |
| | 593/602 | 0.23276 | 634/641 | 0.367178 | 610/619 | 0.398895 | 614/619 | 0.934434 |
| Random excursions variant (18) * | 591/602 | 0.931952 | 631/641 | 0.336751 | 609/619 | 0.039384 | 609/619 | 0.357162 |
| | 601/602 | 0.032561 | 637/641 | 0.13091 | 617/619 | 0.851109 | 617/619 | 0.38749 |
| Serial (2) | 996/1000 | 0.281232 | 990/1000 | 0.705466 | 994/1000 | 0.542228 | 992/1000 | 0.536163 |
| | 992/1000 | 0.047785 | 991/1000 | 0.183547 | 994/1000 | 0.440975 | 988/1000 | 0.937919 |
| Linear complexity (1) | 996/1000 | 0.711601 | 988/1000 | 0.342451 | 990/1000 | 0.971006 | 992/1000 | 0.919131 |

*) Since this test consists of a large number of subtests, only the smallest and the largest values of the pass rate are reported herein altogether with the corresponding PoP-values.

*Hardware footprint*. An important figure of merit characterizing the new design is its logic silicon real estate. As shown in Chapter 7.1, the proposed hash function requires *h* + *b* FFs, where *h* and *b* are sizes of HRG and NSL, respectively. Moreover, logic gates are also needed to implement all linear and nonlinear functions, data injectors, reset logic, etc. Their quantity depends primarily on *b*. Table 7.4 reports the silicon real estate taken up by different instances of $H_2B$ in terms of gate equivalents (GE), where one GE is the area occupied by a 2-input NAND gate. The presented numbers were obtained with a commercial synthesis tool. All components of logic were synthesized using a 7nm CMOS standard cell library. In addition to clock tree synthesis, the deployed design flow was comprised of placement and route with intermediate timing optimizations. For selected hash functions of Table 7.2, Table 7.4 reports the number of combinational and sequential cells, the number of buffers/inverters, and the number of references, i.e., library components used in building larger blocks. The corresponding GE numbers with respect to the aforementioned cells are listed in the right part of the same table. Finally, the last two columns of the table provide a maximal operating frequency of each design after assessing its critical paths and results of statistical power analysis based on the circuits' switching activity at 400 MHz.

Table 7.4 Hardware footprint - equivalent 2-input NAND gates.

| ID | Size HRG / NSL | Nets | Combina-tional cells | Sequential cells | Buffers inverters | References | Combina-tional area | Buf/inv area | Sequential area | Total area | Speed [GHz] | Power [μW] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 32 / 32 | 348 | 154 | 64 | 37 | 7 | 270 | 28 | 352 | 622 | 6.25 | 89.26 |
| 4 | 32 / 64 | 546 | 256 | 96 | 72 | 7 | 429 | 54 | 528 | 957 | 6.25 | 134.26 |
| 7 | 32 / 96 | 769 | 385 | 128 | 137 | 7 | 614 | 111 | 704 | 1318 | 5.56 | 309.25 |
| 10 | 32 / 128 | 978 | 497 | 160 | 181 | 7 | 783 | 146 | 880 | 1663 | 5.56 | 387.48 |
| 13 | 32 / 192 | 1537 | 767 | 224 | 260 | 17 | 1215 | 209 | 1232 | 2447 | 5.56 | 566.24 |
| 16 | 32 / 256 | 1940 | 974 | 288 | 339 | 17 | 1533 | 273 | 1584 | 3117 | 5.56 | 720.52 |
| 19 | 64 / 320 | 2567 | 1263 | 384 | 427 | 25 | 2015 | 344 | 2112 | 4128 | 5.56 | 958.24 |
| 22 | 64 / 384 | 2979 | 1484 | 448 | 505 | 25 | 2358 | 406 | 2464 | 4822 | 5.56 | 1116.95 |
| 25 | 64 / 448 | 3373 | 1686 | 512 | 589 | 25 | 2665 | 473 | 2816 | 5481 | 5.56 | 1265.88 |
| 28 | 64 / 512 | 3773 | 1895 | 576 | 669 | 25 | 2986 | 537 | 3168 | 6155 | 5.56 | 1423.48 |
| 31 | 96 / 640 | 4998 | 2480 | 736 | 839 | 45 | 3941 | 674 | 4049 | 7989 | 5.56 | 1835.09 |
| 34 | 96 / 768 | 5820 | 2913 | 864 | 1000 | 45 | 4607 | 803 | 4753 | 9360 | 5.56 | 2153.10 |
| 37 | 128 / 896 | 6994 | 3484 | 1024 | 1170 | 61 | 5534 | 940 | 5633 | 11167 | 5.56 | 2542.66 |
| 40 | 128 / 1024 | 7787 | 3898 | 1152 | 1337 | 61 | 6165 | 1073 | 6337 | 12501 | 5.56 | 2837.17 |

## 7.3 Resilience against attacks

Cryptographic hash functions, essential parts of many security systems, can be exposed to various types of malicious activities. This chapter summarizes various arguments applicable to the resilience of the proposed family of hash functions against several forms of attacks and cryptanalysis techniques when $H_2B$ is employed in its unkeyed form, i.e., as a modification detection code (MDC). Typically, attacks on a hash function are defined as algorithms that try to find either a collision, second preimage, or preimage with an adversary having full control over all input bits.

*Collision resistance.* It is an intuitive concept which is rather difficult to formalize [156], as collision attacks depend on particulars of a given hash function [87]. Nevertheless, the previous chapter has shown the number of hash compressions before a collision occurs on the $b$-bit output of the presented function, i.e., on the outputs of NSL. Experiments run for $b = 32, \ldots, 64$ bits confirmed that in all test cases a mean value of this number was greater than the birthday bound. One may conclude, therefore, that the collision resistance level of $H_2B$ is no less than $2^{b/2}$.

*Second preimage resistance.* Recall that collision resistance is strictly stronger than second preimage resistance [94], and thus collision resistance implies second preimage resistance (also referred to as weak collision resistance). As a result, the second preimage resistance of $H_2B$ is also expected to be $2^{b/2}$.

*Preimage resistance.* Up to date, it is impossible to find any systematic (algorithmic) method that could be used to reverse processing steps of $H_2B$, hence to find an input sequence leading to a given state of NSL. Since $H_2B$ consists of the hybrid linear ring generator feeding the sequential block deploying a group of highly nonlinear yet balanced Boolean functions in its feedback network, it makes linearizing all rounds of $H_2B$ but a very few computationally infeasible. In other words, finding such an input has the complexity of a brute force search that would take approximately $2^b$ steps. As the proposed scheme is not constrained by the output size $b$ and the number of rounds – they can be freely selected as shown in the previous chapters – $H_2B$ can be regarded secure.

*Differential attacks.* They exploit nonuniform propagation of differences within blocks forming a given cryptographic hash function. To analyze the resistance of $H_2B$ to differential attacks, experiments were run for all designs of Table 7.2 similar to those of the avalanche tests and aimed at checking a responsiveness of successive output bits to a single input change. Given a random binary token $r$, test flips a given bit of $r$ to obtain token $r'$, determines the corresponding digests $d$ and $d'$, and then $D = d \oplus d'$. The value of D is subsequently used to update H($k$) (again, H is a $b$-bin histogram, where $b$ is the digest size) that counts the number of times the output bit $k$ has flipped in a response to a single-bit change in the input. After repeating this experiment N times, one can check the average value of H($k$) for $k = 0, \ldots, b - 1$ altogether with minimal and maximal results. The same experiment has to iterate for all or a subset of input bits. The actual experiments were run for N = 10,000 and by flipping all input bits. In all examined test cases, the obtained results have confirmed that the resultant hash values cannot be distinguished from random samples, and therefore an adversary will be unable to detect statistical patterns in the distribution of D's. For the sake of illustration, consider the results obtained for design no. 5 with $b = 64$ and by flipping bits of each 63-bit token (see Table 7.5). For each input bit, the table reports the mean value of H($k$), as well as the minimal and maximal values of H($k$) that could be indicative of any excessive

Table 7.5 Differential attack results.

| Input | Mean | Min | Max | Input | Mean | Min | Max | Input | Mean | Min | Max |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.500013 | 0.498943 | 0.501206 | 21 | 0.499937 | 0.499008 | 0.500875 | 42 | 0.500046 | 0.499091 | 0.500865 |
| 1 | 0.50015 | 0.49909 | 0.501567 | 22 | 0.500126 | 0.499129 | 0.501354 | 43 | 0.500029 | 0.499038 | 0.500623 |
| 2 | 0.499936 | 0.498999 | 0.50093 | 23 | 0.49997 | 0.499114 | 0.500878 | 44 | 0.499924 | 0.4991 | 0.500727 |
| 3 | 0.500062 | 0.499256 | 0.501063 | 24 | 0.499999 | 0.498861 | 0.501042 | 45 | 0.500018 | 0.498695 | 0.50101 |
| 4 | 0.500098 | 0.499298 | 0.50126 | 25 | 0.500086 | 0.499438 | 0.500892 | 46 | 0.500014 | 0.498984 | 0.501057 |
| 5 | 0.499778 | 0.498832 | 0.500587 | 26 | 0.500165 | 0.499348 | 0.501024 | 47 | 0.499962 | 0.499155 | 0.501035 |
| 6 | 0.500044 | 0.49907 | 0.500832 | 27 | 0.499962 | 0.498922 | 0.500641 | 48 | 0.499975 | 0.498955 | 0.500895 |
| 7 | 0.500026 | 0.498195 | 0.501149 | 28 | 0.500018 | 0.499291 | 0.50101 | 49 | 0.500075 | 0.498592 | 0.501307 |
| 8 | 0.499929 | 0.49912 | 0.500564 | 29 | 0.49979 | 0.498972 | 0.500976 | 50 | 0.499833 | 0.499019 | 0.501219 |
| 9 | 0.49989 | 0.498752 | 0.500966 | 30 | 0.500144 | 0.499204 | 0.501375 | 51 | 0.499888 | 0.498814 | 0.500975 |
| 10 | 0.499934 | 0.498958 | 0.501399 | 31 | 0.499941 | 0.498818 | 0.500785 | 52 | 0.499828 | 0.498791 | 0.500901 |
| 11 | 0.499885 | 0.499044 | 0.501367 | 32 | 0.49994 | 0.499213 | 0.500685 | 53 | 0.500045 | 0.498924 | 0.500946 |
| 12 | 0.499973 | 0.498752 | 0.501033 | 33 | 0.500045 | 0.49922 | 0.501242 | 54 | 0.499813 | 0.499107 | 0.500469 |
| 13 | 0.500028 | 0.499056 | 0.50056 | 34 | 0.499904 | 0.499174 | 0.500739 | 55 | 0.500007 | 0.499135 | 0.501291 |
| 14 | 0.499949 | 0.499389 | 0.500858 | 35 | 0.499894 | 0.498804 | 0.500692 | 56 | 0.500085 | 0.499218 | 0.500831 |
| 15 | 0.500021 | 0.499062 | 0.500944 | 36 | 0.500018 | 0.498931 | 0.501218 | 57 | 0.500081 | 0.498574 | 0.501422 |
| 16 | 0.499984 | 0.498651 | 0.500761 | 37 | 0.500006 | 0.498414 | 0.500836 | 58 | 0.499903 | 0.498834 | 0.50108 |
| 17 | 0.500001 | 0.49879 | 0.50095 | 38 | 0.500099 | 0.499352 | 0.501236 | 59 | 0.499916 | 0.498739 | 0.501195 |
| 18 | 0.500056 | 0.498308 | 0.501083 | 39 | 0.499998 | 0.498912 | 0.501002 | 60 | 0.499892 | 0.498646 | 0.500771 |
| 19 | 0.500177 | 0.499484 | 0.501894 | 40 | 0.499911 | 0.498835 | 0.501051 | 61 | 0.499987 | 0.499161 | 0.501043 |
| 20 | 0.499947 | 0.49896 | 0.500885 | 41 | 0.500065 | 0.499255 | 0.501263 | 62 | 0.50005 | 0.498959 | 0.501059 |

deviations from the expected average. As can be easily verified, for all token bits, the corresponding toggling rates of output bits are very close to 0.5, and this is also confirmed earlier by the statistics (11) computed for the avalanche tests (Chapter 7.2).

*Cube attacks.* A cryptographic hash function can be seen as a black-box computing a set of Boolean functions. It paves the way for algebraic cryptanalysis techniques, including so-called cube attacks which are based on the reformulation of hashing results as polynomial functions over GF(2). The actual attack consists of 1) a preprocessing phase that selects randomly a subset of variables called a maxterm to verify if the resultant factorization of the original function by the maxterm yields a linear expression, and then 2) the online phase where, given a large number of maxterms and linear expressions found earlier, one uses a Gaussian elimination to recover secret variables [53]. This type of attack is applicable to functions whose ANF has low degree and low density. Thus, a symbolic simulation of selected instances of $H_2B$ was executed to reconstruct ANFs on its outputs. These experiments are only feasible for a relatively short s-bit input messages and small registers of HRG and NSL (the latter items determine the number of secret variables). Table 7.6 illustrates a basic trend for $s = 16$, $b = 32$, and four 5-input nonlinear functions deployed as a feedback network of NSL. Given an output bit, the corresponding entries to the table show how many monomials of a given degree $v \in [s - 3, s]$ occur in ANF of a function associated with that bit.

Table 7.6 The number of monomials of a given degree.

| Bit | s – 3 | s – 2 | s – 1 | s | Bit | s – 3 | s – 2 | s – 1 | s |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 265 | 64 | 7 | 1 | 16 | 295 | 58 | 8 | 1 |
| 1 | 260 | 52 | 10 | 0 | 17 | 276 | 62 | 7 | 0 |
| 2 | 268 | 52 | 9 | 0 | 18 | 283 | 62 | 7 | 1 |
| 3 | 282 | 61 | 10 | 1 | 19 | 282 | 64 | 4 | 0 |
| 4 | 285 | 60 | 9 | 0 | 20 | 295 | 66 | 7 | 0 |
| 5 | 288 | 58 | 9 | 1 | 21 | 291 | 57 | 10 | 0 |
| 6 | 301 | 63 | 11 | 1 | 22 | 285 | 50 | 9 | 0 |
| 7 | 273 | 61 | 8 | 0 | 23 | 264 | 52 | 9 | 0 |
| 8 | 291 | 59 | 6 | 1 | 24 | 279 | 51 | 11 | 0 |
| 9 | 268 | 57 | 7 | 1 | 25 | 298 | 65 | 7 | 1 |
| 10 | 267 | 52 | 7 | 1 | 26 | 285 | 73 | 10 | 0 |
| 11 | 278 | 61 | 11 | 1 | 27 | 276 | 59 | 10 | 1 |
| 12 | 300 | 67 | 10 | 1 | 28 | 267 | 60 | 9 | 1 |
| 13 | 270 | 50 | 6 | 0 | 29 | 268 | 56 | 12 | 1 |
| 14 | 271 | 57 | 8 | 0 | 30 | 302 | 58 | 9 | 0 |

These and other results not reported here clearly confirm a more general observation that if the number n of inputs tends to infinity, random Boolean functions have almost certainly algebraic degree at least $n - 1$ since the number of Boolean functions of algebraic degree at most $n - 2$ is negligible with respect to the number of all $n$-argument Boolean functions [27].

The above findings are necessary to conclude that $H_2B$ is virtually immune to a cube attack. However, even a high algebraic degree of a hash function may not guarantee that all usable maxterms be of similar degree. Indeed, several maxterms can be of degrees which are considerably lower than the degree of the entire function. This is because a cube attack sets to zero all public variables that are not part of a maxterm, thus eliminating many higher degree terms. In fact, the degree of usable maxterms depends primarily on the ratio of the number of secret variables to the number of public variables and the extent to which the secret variables have diffused throughout the function. Fortunately, the high degrees of the resultant functions associated with all outputs of NSL (see Table 7.6) ensure that monomials of degree at least $s + 1$ comprise the secret variables.

*Slide attacks.* A requirement for a slide attack to work on a cryptographic hash function is that it can be broken down into multiple rounds of an identical function. As shown in the previous chapters, $H_2B$ makes it impossible to have two valid initial states shifted by a certain number of clock cycles, and such that the shift persists through all iterations required to complete the hashing process. As a result, the proposed hash function is resistant against slide attacks.

*Side-channel attacks.* This class of attacks comprises several methods aimed at extracting information from microelectronic devices through analyzing their physically observable characteristics such as power consumption, processing time, electro-magnetic radiation, heat dissipation, or light and acoustic emissions. Interestingly, the proposed hash function has approximately half of its flip-flops set to 1 most of the time. Hence, it is virtually impossible to trace and interpret its internal states at any processing cycle by means of external leakage probes. For example, a power consumption model aimed at detecting the number of transitions in each cycle has to take account of toggling in both HRG and NSL. However, with secret keys initializing both registers to patterns having roughly the same counts of 0s and 1s, and with nonlinear functionality of NSL, such analysis becomes impractical.

In contrast to multiple-round-based hash functions that feature separate steps such as substitutions, row shifts, or column swaps, $H_2B$-based hashing is carried out simultaneously within its linear and nonlinear logic. This modus operandi makes it infeasible to derive useful internal values or sequential states, and it is hard to trace any HRG or NSL-specific characteristics. Finally, one may observe that $H_2B$ was primarily designed as a key part of an embedded hardware root of trust. As such, it is hard to measure any physical characteristic of either $H_2B$ or the root of trust as a whole.

## 7.4 Built-in self-test

As the root-of-trust mission is to secure ICs and to protect their design-for-test (DFT) infrastructure, its test should be an autonomous routine that relies either entirely or in large part on internal on-chip resources that do not interfere with other DFT components such as scan chains. As logic BIST provides neither full observability nor full controllability of internal storage elements from the IC interface, it can be used to test hardware of cryptographic hash functions such as $H_2B$. In LBIST, the original circuit is typically enhanced by PRPGs and test response compactors. The simplicity and functionality of the new design, however, can facilitate its self-testing without resorting to additional test logic blocks.

The entire LBIST session is based on the $H_2B$ native functionality. In particular, HRG is repurposed to be used as a PRPG. It is worth noting that pseudorandom data and possible errors can easily propagate through the HRGs due to their functionality. Moreover, NSL can act as a MISR producing a final test result. It observes outputs of the phase shifter and nonlinear functions, and accumulates test responses.

A test session begins by resetting all memory elements comprising $H_2B$ except one stage of HRG which is set to 1. Similarly to the mission mode, the actual test runs for a number of clock cycles necessary to produce a digest. After test completion, valid test results are available in the memory elements of NSL. This simple BIST procedure is fault simulated for all single stuck-at faults within the circuitry every $H_2B$ of Table 7.2 consists of. It appears that $4b$ clock cycles suffice to obtain a complete coverage of all testable stuck-at faults; recall that $b$ is the number of FFs the NSL block consists of. Interestingly, no aliasing events were observed, i.e., having a signature of a fault-free circuit produced by a faulty device.

A slightly different BIST scenario has to be used to test programmable instances of $H_2B$. In order to make it testable, HRG is redesigned in such a way that it becomes capable of generating autonomously tests that subsequently will feed the selection mask register. It can be accomplished by virtue of a 2-bit twisted ring counter that reuses two rightmost FFs of HRG, as shown in Fig. 7.3. Regardless of how the counter is initialized, it will go through the following sequence of states: 00 $\rightarrow$10 $\rightarrow$ 11 $\rightarrow$ 01 $\rightarrow$ 00, thus producing on its output a repeatable pattern 0011 – the test sequence which suffices to check the integrity of the mask register and the logic it drives. A multiplexer is used to interface the counter with the mask register, with no need for any initialization circuitry. Again, the proposed circuitry was fault simulated to find out that $4b$ clock cycles  are enough to ensure that all nets in the circuit assume both values: 0 and 1. It allows one to excite all stuck-at-1 and stuck-at-0 faults, respectively.

## 7.5 Comparison with other schemes

Given the volume of the earlier work, it is virtually infeasible to offer a compact yet comprehensive comparison of $H_2B$ with the remaining state-of-the-art solutions. Especially since such a comparison or assessment might be misleading or unfair for several reasons. For example, there is always a certain degree of uncertainty as far as the actual implementation of any hash function is concerned, including the impact of a deployed semiconductor technology. Moreover, existing solutions differ in terms of design assumptions, their primary objectives, as well as metrics used to evaluate particular schemes. Fortunately, the work presented recently in [191] provides a meticulous comparison of lightweight hash functions in terms of their performance. One may find there details of a silicon real estate, throughput, latency, and power consumption of 34 lightweight hash functions used in academia and industry.

Based on the results presented in [191] and also [6], Table 7.7 shows the aforementioned data regarding a few state-of-the-art hashing schemes (references regarding those schemes can be found in [191]; the numbers following the name give the digest size). The second column of the table provides the silicon area used by the hash functions in terms of 2-input NAND gates, similarly to a metric deployed in Chapter 7.2. The next column reports the number of clock cycles necessary to arrive with the final hash value. Column Throughput gives the number of Kbits per seconds that corresponds to the number of plaintexts processed per a time unit. Finally, the last column quantifies the amount of power (in $\propto$W) required to run the hashing circuitry. It is worth noting that throughput and power metrics were obtained for a frequency of 100 KHz. For the sake of comparison, the same table offers the extracts of earlier data characterizing a few instances of $H_2B$ (recall that these data were obtained for 400 MHz frequency). As can be seen, given the digest size, the proposed solution deploys similar gate counts and needs iterations in the same ranges as those of other schemes introduced earlier in the technical literature and in the industry. At the same time, thanks to its shall combinational logic, performance and power consumption of $H_2B$ either compares favorably or remains similar relative to its much higher working frequency.

Table 7.7 Comparison with other hash functions.

| Hash function | Area [GE] | Latency [cycles] | Through-put [Kbps] | Power [µW] |
|---|---|---|---|---|
| ARMADILLO-80 | 2923 | 176 | 272.00 | 44.00 |
| ARMADILLO-160 | 5406 | 320 | 250.00 | 83.00 |
| KECCAK-128 | 2520 | 900 | 8.00 | 5.60 |
| KECCAK-160 | 4900 | 900 | 222.22 | 27.60 |
| PHOTON-128 | 1122 | 996 | 1.61 | 2.29 |
| PHOTON-224 | 2786 | 204 | 15.69 | 6.50 |
| DM-PRESENT-80 | 1600 | 547 | 14.63 | 1.83 |
| H-PRESENT-128 | 4256 | 32 | 200.00 | 8.09 |
| U-QUARK-136 | 1379 | 544 | 1.47 | 2.44 |
| S-QUARK-256 | 2296 | 1024 | 3.13 | 4.35 |
| SLISCP-hash-160 | 2492 | 144 | 29.62 | 7.44 |
| SLISCP-light-160 | 2051 | 96 | 44.44 | 5.05 |
| SPONGENT-128 | 1060 | 2380 | 0.34 | 2.20 |
| SPONGENT-256 | 3281 | 120 | 11.43 | 6.62 |
| $H_2B$-128 (10) | 1663 | 512 | 200 Mbps | 387.48 |
| $H_2B$-256 (16) | 3117 | 1024 | 200 Mbps | 720.52 |
| $H_2B$-512 (28) | 6155 | 2048 | 200 Mbps | 1423.48 |

# 8. Stream cipher for scan encryption

Additive SCs play a host of roles in securing variety of digital ecosystems. In particular, they can encrypt and decrypt test data used in manufacturing and in-system test of digital ICs. By assuming such a role, SCs become an essential part of hardware roots of trust that protect ICs against hardware security threats. However, a common concern raised by IC vendors is related to the complexity of security primitives that are not optimized for their hardware implementations. This concerns had motivated the following solution that introduces a new lightweight and scalable test data stream cipher. It combines a HRG with two nonlinear Galois feedback shift registers to yield a number of parallel, cryptographically secure pseudorandom keystreams.

## 8.1 General architecture

The cipher presented in this chapter is primarily destined to be a part of DFT ecosystems where it can work with a RoT handling secure operations of on-chip test logic used by both the semiconductor manufacturing test and in-system test solutions. For example, it can be used as a part of a DFT authentication protocol within an SSN. It places two ciphers on inputs and outputs of the SSN bus and the IJTAG network, as shown in Fig. 8.1, in order to decrypt and encrypt the content of the IJTAG communication and the SSN bus. These ciphers must be very fast – they have to match the speed of SSN typically operating at much higher shift frequencies than in-core DFT logic does.

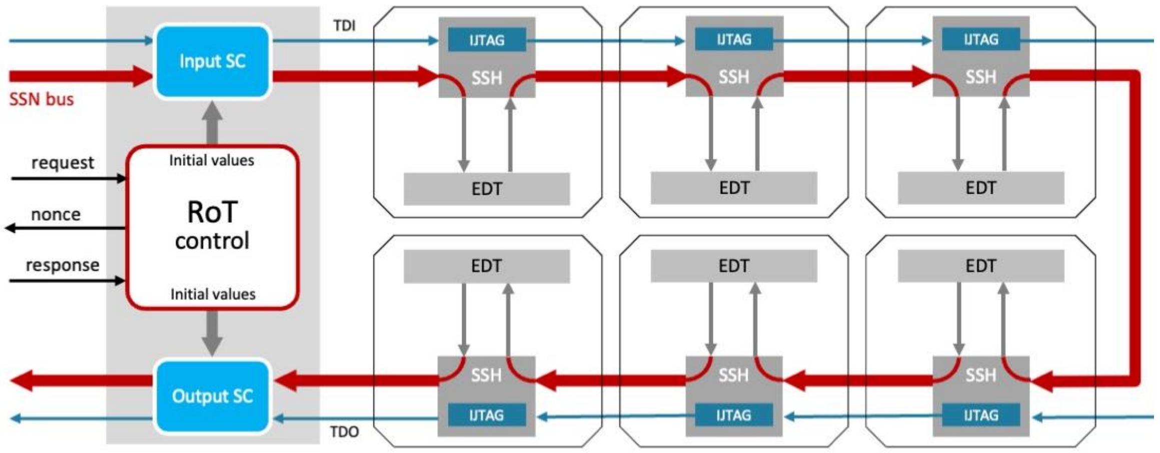


Figure 8.1 Root of trust protecting a 6-core SoC design using SSN [40].

Although both ciphers use the same principles, they can differ with respect to architectural details of modules they both consist of. The proposed SC is comprised of three building blocks: an $h$-bit maximum-length programmable hybrid ring generator (PHRG) and two maximum-length NLFSRs [66], which are $m$- and $n$-bit wide; Fig. 8.2 is a block diagram of this design. In general, the values of $h$, $m$, and $n$ can be pairwise different, and the periods of these registers are $2^h - 1$, $2^m - 1$, and $2^n - 1$, respectively. The sequential parts are initialized by uploading secret values, including a selection mask that sets a feedback function of PHRG. To maintain a high diffusion level of the cipher, the selection mask is dynamically retrieved and stored in a mask register (MR) as a part of PHRG initialization process, as explained in Chapter 8.2. In a mission mode, NLFSR-produced streams pass through linear filters to obtain signals among which are those that PHRG randomly selects to form pseudorandom keystreams. The distinctive feature of the proposed design is its selection mechanism that allows PHRG to pick certain signals and output successive bits of a keystream. This is accomplished through selectors – represented by green boxes in Fig. 8.2 – one per a single output. Keystreams are finally used to yield encrypted data by following the Vernam principle.

A magnified green box in Fig. 8.2 illustrates how a single selector is designed. It employs two $v$-input multiplexers implementing a "one-hot" selection method. The multiplexers receive dedicated control signals from PHRG via its phase shifter [121]. The latter device is employed to enlarge the number of shifted replicas of a PHRG-specific $m$-sequence (recall that PHRG is a customized form of a maximal linear finite state machine). Effectively, the selector outputs a sum modulo 2 of two bits, each of which is picked randomly from one of $v$ linearly filtered NLFSR-produced sequences. As both multiplexers accept $v$ data inputs, the scheme enables $v^2$ dynamic pairs of NLFSR streams in total to produce a single keystream, one pair (combination) per clock cycle. For example, if $v = 4$, then every selector of Fig. 8.2 allows one to combine two nonlinear streams at a time in one of 16 different ways.

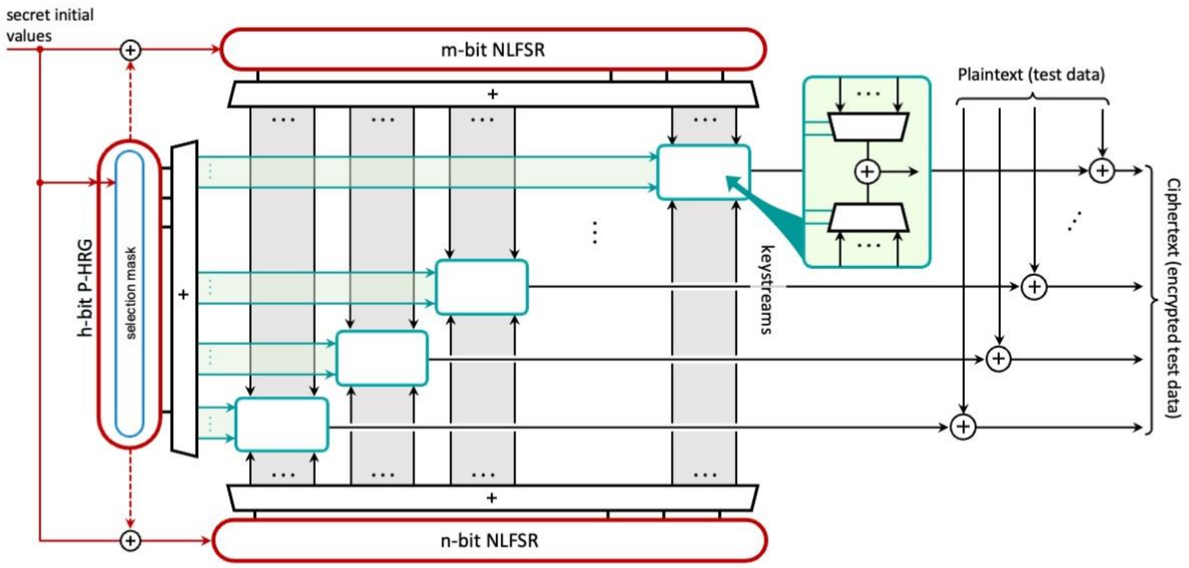


Figure 8.2 Block diagram of the proposed SC.

## 8.2 Programmable hybrid ring generators

PHRGs allows one to shift-in a secret mask that sets and retains the feedback function when this unit is in operation. To initialize the entire circuitry, a selection mask and an initial value, both encoded, are separately uploaded to MR and PHRG, respectively, using an XOR gate (Fig. 8.3). When MR is receiving data, clocking of PHRG is disabled and vice versa. Next, the two sequences are blended with each other by running both registers in parallel until MR reaches a state representing a desired feedback function and PHRG settles down to a predefined initial state. During that phase MR receives data from PHRG while the resultant content of two selected FFs of PHRG is superposed with two secret keys initializing both NLSFRs, as shown by the dotted lines in Fig. 8.2. Since then, the MR clocking and links between PHRG and NLFSRs are all disabled.

The input sequence used by the above process is obtained by virtue of O($h$) backward logic simulation of both registers, beginning with a final selection mask and a PHRG state (see Fig. 8.4a; Boolean variables $a$, …, $g$ represent here the PHRG content but one FF set to 1). In principle, this technique reverses direction of all PHRG connections but feedback taps

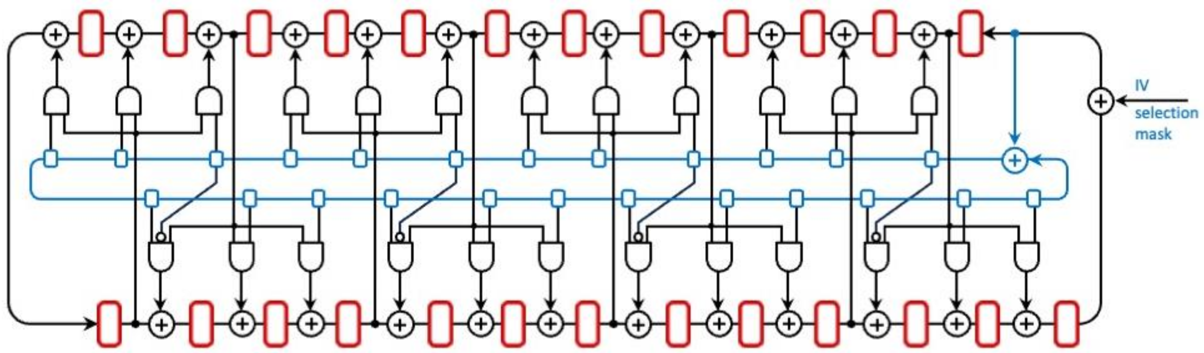


Figure 8.3 Programmable 25-bit HRG.

whose masks show up gradually in MR as initialization progresses. It allows one to run a given PHRG backwards provided there is a single PHRG FF that is not controlled by MR. Then its predecessor can feed MR and make it possible to easily recover a previous bit of the mask, and thus a feedback function, to carry PHRG another step backwards. Fig. 8.4b is a single step back of MR fed by a logic 1 from PHRG, while Fig. 8.4c shows the resultant step back of PHRG itself. Note that certain FFs of MR control two feedback taps in a mutually exclusive manner. This is to avoid having two oppositely disposed feedback nets, not separated by any HRG FF; such nets make a given HRG irreversible as certain states would be reachable from two predecessors.

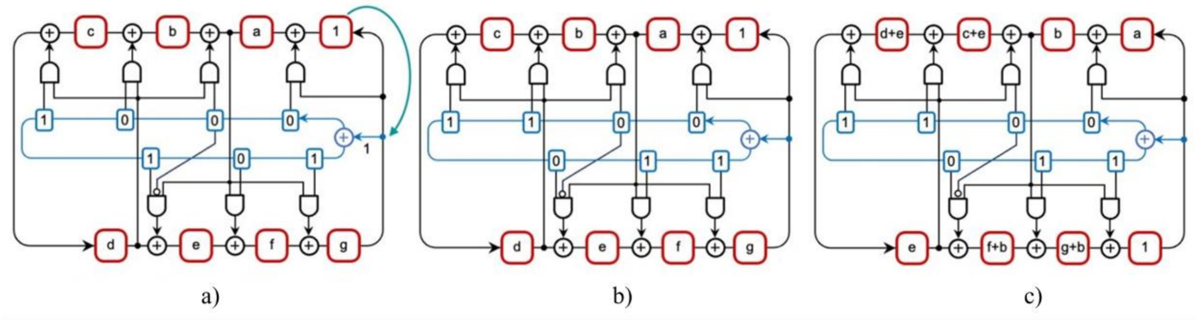


Figure 8.4 Backward simulation of the 8-bit PHRG: a) a desired initial state (a simulation starting point), b) registers after a single step back of MR, and c) after a single step back of PHRG.

A direct use of HRG to control the selectors of Fig. 8.2 may compromise the encryption quality because of structural and linear dependencies between the HRG outputs. To reduce such dependencies, a phase shifter is deployed in conjunction with HRG. It entails a set of linear combinations of the HRG outputs. Obtained sequences are shifted with respect to every other sequence by at least a predefined number of bits. Furthermore, a phase shifter can enlarge the HRG output space, that is, it allows a relatively short HRG to drive a large number of selectors. It may substantially reduce the cipher's sequential logic footprint, as shown in Section 8.4G.

## 8.3 Primitive nonlinear feedback shift registers

As shown in Fig. 8.2, the new SC employs two maximum-length (primitive) NLFSRs whose output sequences are typically of very large linear complexity and high degrees of security, and thus immune to, for example, algebraic attacks. Recall that an $n$-bit primitive NLFSR has a period of $2^n - 1$ (inclusion of a missing state requires adding a circuit driven by all $n$ variables; usually such a solution does not justify this incremental enhancement). Both NLFSRs are of Galois type, where, in principle, every FF can be updated by its own individual feedback function [59]. The actual structure of NLFSRs adapted here is similar to the example shown in Fig. 8.5. As in HRG, an $n$-bit NLFSR consists of an $n$-bit circular shift register and a number of feedback taps driven by Boolean functions of up to five variables.

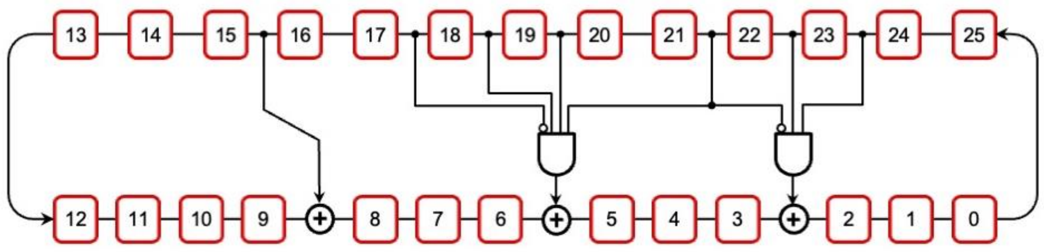


Figure 8.5 26-bit primitive NLFSR with ANF 0, 7, (20, 21), (13, 14, 16), (19, 20, 21), (12, 13, 14, 16).

Clearly, at least one of these functions has to be nonlinear. To make the used NLFSRs layout-friendly and capable of matching the SSN speed, their structure is further optimized in such a way that each feedback function is fed by five, preferably consecutive, FFs at the most, and these input variables are constrained to the “upper” section of the register, if possible.

Back to Fig. 8.5, it illustrates a primitive 26-bit NLSFR, where three FFs (2, 5, and 8) are driven, through the XOR gates, by the following functions:

$$f_2 = {\sim}x_{22} \cdot x_{23} \cdot x_{24}, \quad f_5 = {\sim}x_{18} \cdot x_{19} \cdot x_{20} \cdot x_{22}, \quad f_8 = x_{16},$$

where “~” denotes a logic inversion. The corresponding algebraic normal form of the feedback function is given by:

$$x_0 + x_7 + x_{20} \cdot x_{21} + x_{13} \cdot x_{14} \cdot x_{16} + x_{19} \cdot x_{20} \cdot x_{21} + x_{12} \cdot x_{13} \cdot x_{14} \cdot x_{16}.$$

The above formula can be verified by using a method presented in [58]. Boiling the many details down, the presented structure is first brought back to its equivalent Fibonacci format whose feedback function can be easily retrieved, and then used to obtain the ANF by means of well-known algorithms. We adapt a notation where all variables are replaced with their indexes, and all product terms comprising more than a single variable are grouped by parentheses. For instance, the ANF of NLFSR of Fig. 8.5 can be written down as 0, 7, (20, 21), (13, 14, 16), (19, 20, 21), (12, 13, 14, 16). It is worth noting that this particular NLFSR has a fully planar structure, and thus it is amenable to efficient place-and-route steps that provide layout-friendly and timing-optimized solutions. Fig. 8.6 is another NLFSR example. It is a primitive 28-bit circuit with six all-nonlinear feedback taps. Its ANF is given by 0, 14, 23, 6, (10, 9), (11, 14), (23, 25), (23, 26), (6, 8), (10, 8, 9), (12, 15, 16), (18, 19, 20), (23, 25, 26), (16, 18, 19, 20). In contrast to a direct ANF-based implementation of this NLFSR that would need 10 AND gates and even more 2-input XOR gates, the circuit of Fig. 8.6 is area optimized and employs just six AND gates and six XOR gates, thus achieving a visible reduction of both the gate count and the latency. NLFSRs of Fig. 8.5 and 8.6 have been found, among others, by a search engine described briefly at the end of this chapter.

Primitive $n$-bit Galois NLFSRs are capable of generating output sequences with the period of $2^n - 1$, as shown in [58]. Among them there is always a sequence (and its shifted replicas) that satisfies the first two postulates of randomness by Golomb [71], i.e., (1) in a

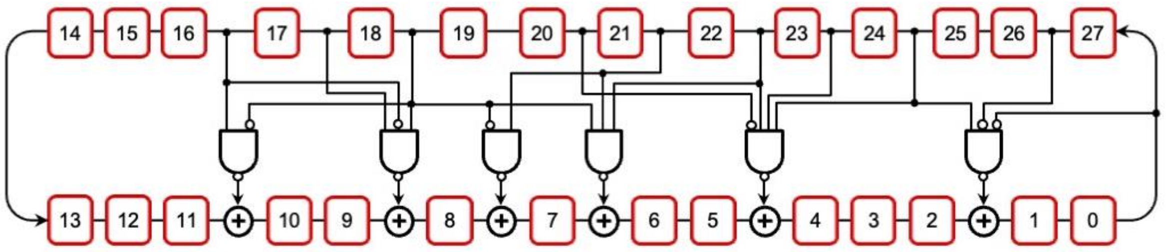


Figure 8.6 28-bit primitive NLFSR with ANF 0, 6, 14, 23, (10, 9), (11, 14), (23, 25), (23, 26), (6, 8), (10, 8, 9), (12, 15, 16), (18, 19, 20), (23, 25, 26), (16, 18, 19, 20).

period of the sequence, the 0 and 1 counts differ by at most 1, (2) in every period, $2^{-k}$ of runs have length $k = 1, 2$ …, as long as the number of runs so indicated is greater than 1. For each of these lengths, there are as many 0-runs as 1-runs. Also, each $n$-tuple occurs in the sequences exactly once but a missing state which is a feedback function dependent. Consider again the NLFSR of Fig. 8.5. Sequences observed on the outputs of FFs 2, 1, 0, 25, 24, …, 9 contain all $2^{26} - 1 = 67\,108\,863$ different (overlapping) 26-tuples. Two other sequences on FFs 6, 7, 8, and 3, 4, 5 have the period of $2^{26} - 1$, as well. They comprise, however, 41 641 431 and 37 963 775 different 26-tuples, respectively. In these two cases, certain 26-tuples occur more than once, whereas others are missing. In general, an $n$-bit Galois NLFSR with $m$ feedback taps yields as many as $m$ different sequences (and their shifted replicas) having the period of $2^n - 1$, where $m$ is also the number of groups of consecutive FFs separated by $m$ XOR gates. The number of different $n$-tuples produced by each output can be used to rank them and thus to help the process of selecting sequences that will feed an associated linear filter (Fig. 8.2).

To increase the number of sequences that drive the multiplexers (see Fig. 8.2), outputs of both NLFSRs are linearly filtered within circuits referred to as the *expanders*, made of XOR gates whose fan-in is either two or three to reduce expanders hardware footprint. Although linear functions that take as inputs at least two stages of a given $n$-bit primitive NLFSR yield a sequence with a period of $2^n - 1$ [67], little is known about the distribution of overlapping $k$-tuples across a single period of such a sequence. The number of different $n$-tuples in the same sequence remains an open problem, too. For example, three different output sequences observed in the NLFSR of Fig. 8.5 (as mentioned earlier) give rise to 70 new sequences with the period of $2^{26} - 1$ produced by 2-input XOR gates, and additional 783 sequences of the same period obtained by virtue of 3-input XOR gates. Sequences with the highest counts of overlapping 26-tuples are formed by the following linear filters: $x_2 + x_9$ (65 174 655 overlapping 26-tuples), $x_0 + x_1$ (56 623 103), $x_0 + x_{24}$ (51 445 759), $x_0 + x_{21}$ (49 082 367), $x_0 + x_{23} + x_{24}$ (48 529 407), and $x_0 + x_{20} + x_{24}$ (48 369 663).

A single keystream is finally generated by XOR-ing two nonlinear sequences observed on the outputs of the corresponding multiplexers. As shown before, these devices are fed by the expander outputs, i.e., linear combinations of NLFSR stages. To avoid reconvergence of a signal originating at a certain NLFSR stage, each multiplexer accepts only those expander outputs that do not share NLFSR-produced sequences. Therefore, a multiplexer may have inputs such as $x_2 + x_9$ and $x_3 + x_7$, but it will avoid, for instance, inputs $x_0 + x_1$ and $x_0 + x_7$. A more detailed characterization of the keystreams is provided in Chapter 8.4.

A final note on how the deployed NLFSRs were actually identified. No practical way is known to determine a feedback function of an $n$-bit NLFSR that leads to a maximum-length period of $2^n - 1$. Although the scheme of [56] allows designers to construct primitive Galois NLFSRs for large $n$ in a time-efficient manner, a linear complexity of the resultant sequences is low and such NLFSRs do not achieve a desired cryptographic security. This is because the proposed approach reorders partially a state trajectory of a primitive $n$-bit LFSR by adding two copies of a nonlinear Boolean function to the LFSR feedback network; these functions are then moved apart from each other. Since checking whether other $n$-bit NLFSRs are primitive takes $O(2^n)$ CPU time, experiments employed an FPGA-based search engine comprising a large number of identical modules operating in parallel. The chief function of every module is to run a Galois NLFSR, structurally similar to those of Fig. 8.5 and 8.7, and to verify its period. It requires control signals to select a feedback function, and thus to setup an NLFSR it is going to examine. The control part of the engine consists of a large, free-running, LFSR-based control data generator. In response to a request received from an individual module, it returns a pseudorandom sequence of signals that determine a nonlinear feedback function. It consists of several feedback taps, each being a product of up to five input variables.

Consequently, a control sequence includes the number of feedback taps as well as bits that enable both successive taps and successive variables of a given tap. Additional bits indicate, per a feedback tap, whether a given input variable is to be inverted. Finally, there are bits encoding locations of the XOR gates – the outputs of the enabled taps, and signaling which outputs are to be inverted.

Table 8.1 lists selected primitive Galois NLFRSs, ranging in size $n$ from 16 to 32. They were found by using the described FPGA-based search engine. Note that other NLFSRs of size up to 24 are also reported in [57]. The second column of the table uses a shorthand notation of a feedback function as a list of parenthesized feedback taps (product terms). Every tap consists of a FF ID (receiver:) fed, through an XOR gate, by an appropriate product of variables and their complements; here only the index $i$ of the variable $x_i$ is retained, and thus it can be represented by a FF ID (driver). Inverted variables appear with a minus sign. Similarly, if a NAND gate forms a given tap, then a minus precedes the receiver ID. For example, the NLFSR of Fig. 8.5 is defined as follows: (2: 24, 23, -22), (5: 22, 20, 19, -18), (8: 16). The next three columns give the number of different (not shifted replicas) output sequences with the period of $2^n - 1$, which are either observed directly on the outputs of a given NLFSR (column S) or are obtained by using a 2-input XOR gate driven by certain outputs of the NLFSR (column D), as well as those obtained by a similar linear filter having three inputs (column T). The linear complexity of sequences comprising all $2^n - 1$ $n$-tuples is given in column LC for $n \leq 22$. The last two columns detail a few selected output sequences with the period of $2^n - 1$ but having the number of different $n$-tuples smaller than $2^n - 1$. First, the column XOR provides FFs driving a given XOR gate (a linear filter) producing the sequence whose $n$-tuple count is reported in the subsequent column. Consider, for instance, the value of 890 999 listed for $n = 20$. It indicates that a 3-input XOR gate fed by FFs 0, 1, and 19 yields a sequence that features 890 999 different 20-bit tuples (out of $2^{20} - 1$ possible combinations).

## 8.4 Experimental results

The presented SC was validated by means of several tests [24], including two statistical test suites [12], [179] from the NIST and another test set (AIS-31) provided by the German IT security certification authority (BSI) [91]. This chapter reports the results obtained for four instances of the cipher with the following values of *h*, *m*, and *n* (the last value *c* gives the total number of states a given cipher can cycle through):

- $L_1$: $h = 48$, $m = 17$, $n = 19$, $c \approx 1.93 \times 10^{25}$,
- $L_2$: $h = 61$, $m = 31$, $n = 32$, $c \approx 2.13 \times 10^{37}$,
- $L_3$: $h = 89$, $m = 31$, $n = 32$, $c \approx 5.71 \times 10^{45}$,
- $L_4$: $h = 127$, $m = 31$, $n = 32$, $c \approx 1.57 \times 10^{57}$.

The architectural details of the maximum-length NLFSRs used in the experiments can be found in Table 8.1. The maximum-length feedback functions of the PHRG are as follows:

- $f_1(x) = x^{48} - x^{41} - x^{35} - x^{28} + x^{22} - x^{17} - x^{12} + x^{8} + 1$,
- $f_2(x) = x^{61} - x^{53} + x^{45} + x^{38} - x^{29} + x^{20} + x^{14} - x^{5} + 1$,
- $f_3(x) = x^{89} + x^{80} - x^{70} + x^{58} - x^{46} - x^{34} - x^{23} + x^{10} + 1$,
- $f_4(x) = x^{127} - x^{108} - x^{90} + x^{75} + x^{64} - x^{46} - x^{34} + x^{16} + 1$.

Moreover, SCs in all cases used 64 selectors, each comprising two 2-input multiplexers to yield 64 keystreams, as shown in Fig. 8.2. Due to space constraints, the tables only report the results for a single keystream of each cipher except Table 8.2.

Binary sequences (keystreams) subjected to tests were serially collected from all 64 outputs. To mimic accurately the cipher behavior, all its registers were reseeded every $c = 10^6$ clock cycles with pseudorandom values produced by the Mersenne Twister. The cipher was run for $10^9$ cycles, thereby producing, on each output, N = 1000 consecutive sequences, each comprising $10^6$ bits to meet the requirements of the NIST tests. The same sequences were used to run most of the remaining tests. The next sections introduce each test and discuss the corresponding experimental results. Several tests that have already been described in Chapter 7.2 are also included for the sake of completeness.

### *A. Probability of bit values*

This test is aimed at checking if the logic value of 1 occurs on every bit of a keystream approximately half of the time. Let $P_1(b)$ be the probability of having 1 on bit *b*:

$$P_1(b) = C_b / N, \tag{8.1}$$

where $C_b$ is the 1s count on bit *b*. The sample mean S of $P_1(b)$, $b = 0, 1, \ldots, c - 1$, is then approximately normally distributed under the null hypothesis that the $P_1(b)$'s are independent and identically normally distributed random variables. If $\sigma$ is the sample standard deviation, then the test statistic

$$Z = \sqrt{c}(S - 0.5) / \sigma \tag{8.2}$$

Table 8.1 Selected maximum-length nonlinear feedback shift registers.

| *n* | Feedback taps | S | D | T | LC | XOR | *n*-tuples |
|---|---|---|---|---|---|---|---|
| 16 | (0: 14, 15), (1: -13, 14), (2: 12), (4: 10, -11), (5: 9) | 5 | 61 | 349 | $2^{16} – 6$ | {1, 9, 15} | 51711 |
| | | | | | | {0, 15} | 49151 |
| 17 | (0: 15, 0), (1: 13, 14), (3: 12, -14), (4: -11, 12), (6: 10) | 5 | 68 | 437 | $2^{17} – 2$ | {0, 16} | 98303 |
| | | | | | | {1, 16} | 93569 |
| 18 | (1: 16), (3: 13, -14), (5: -10, 11, 12) | 3 | 46 | 332 | $2^{18} – 13$ | {0, 1} | 196607 |
| | | | | | | {0, 13} | 196351 |
| 19 | (1: 16, -17, 18), (2: 14, 15, -16), (4: 12, 13, 14), (6: -10, 11) | 4 | 66 | 514 | $2^{19} – 2$ | {0, 1} | 393215 |
| | | | | | | {2} | 384207 |
| 20 | (0: -19, 0), (1: -16, 18), (3: 15, -16), (4: -13, 15), (6: -12, 13), (7: 11) | 6 | 97 | 748 | $2^{20} – 2$ | {0, 1, 19} | 890999 |
| | | | | | | {1, 19} | 796739 |
| 21 | (0: 19), (2: -18, 19), (4: -15, 17), (5: 14, -15), (7: -11, 12, 13) | 5 | 88 | 739 | $2^{21} – 2$ | {6} | 1581759 |
| | | | | | | {0, 20} | 1572863 |
| 22 | (0: -19, 20), (2: 18, 20), (4: 16, 17, -18), (6: 14, 15), (8: 13) | 5 | 93 | 825 | $> 2^{21}$ | {0, 21} | 3145727 |
| | | | | | | {1, 5, 9} | 2951856 |
| 23 | (1: 18, 19, 20, -22), (4: 15, 17, 18, -19), (7: 14) | 3 | 61 | 591 | $> 2^{21}$ | {2, 8} | 8098171 |
| | | | | | | {0, 1} | 7471103 |
| 24 | (0: 0, 20, -23), (2: -21, 23), (5: -16, 17), (8: 15) | 4 | 84 | 844 | $> 2^{21}$ | {1} | 14897105 |
| | | | | | | {0, 23} | 12582911 |
| 25 | (4: -20, -19), (-5: 21, 20), (7: 18), (9: 15), (10: -12) | 3 | 24 | 276 | $> 2^{21}$ | {5, 20} | 22466751 |
| | | | | | | {5, 19} | 22443165 |
| 26 | (2: 24, 23, -22), (5: 22, 20,19, -18), (8: 16) | 3 | 70 | 783 | $> 2^{21}$ | {2, 9} | 65174655 |
| | | | | | | {0, 1} | 56623103 |
| 27 | (1: 0, 25, -24, 23), (-3: 25, 24, 23, -22), (-4: 24, 22, 21), (5: 23, -22, 21, -20, 19), (-7: 21, -20, 19, -18, 17), (10: 17, 16, -15) | 6 | 141 | 1593 | $> 2^{21}$ | {2} | 118652195 |
| | | | | | | {4} | 105876356 |
| | | | | | | {1, 2} | 104206650 |
| 28 | (-1: -0, -27, 25), (-4: 25, 24, 23, -21), (-6: 23, 22, 19), (-7: 22, -19), (-8: 19, 18, -17), (-10: -19, 17) | 6 | 147 | 1736 | $> 2^{21}$ | {2} | 248058975 |
| | | | | | | {1, 2} | 201970330 |
| | | | | | | {0, 1} | 201326591 |
| 29 | (-1: -0, 27), (-4: 26, 25, 24, 22), (-6: -24, -23, 21, -20), (7: 23, 21, -20, 19), (-8: -22, -21, 19, -18), (-9: 21, -20, 19), (-10: -20, -19, 17), (11: 18, 17, 16, -15) | 8 | 196 | 2324 | $> 2^{21}$ | {11} | 473137553 |
| | | | | | | {2} | 449487871 |
| | | | | | | {0, 1} | 402653183 |
| 30 | (2: -28, -25), (4: -24, 23), (-6: -25, 24, 21), (7: 24), (-9: -22, -21, -18), (11: 20), (12: 18, 17, 15) | 7 | 178 | 2198 | $> 2^{21}$ | {12} | 806130191 |
| | | | | | | {0, 1} | 805306367 |
| | | | | | | {12, 13} | 762932099 |
| 31 | (4: 26), (-6: 26, 28), (9: 21, -23), (-11: 19, 20, -21, 22), (-12: -19, -21), (13: -17, 18, 19), (14: 17, -19) | 7 | 187 | 2424 | $> 2^{21}$ | {14} | 1703443913 |
| | | | | | | {14, 18} | 1623357704 |
| | | | | | | {0, 1} | 1610612735 |
| 32 | (1: 29, 31), (-4: -26, -27), (-6: 23, 26, -27), (8: 23), (-10: -20, 21) | 5 | 143 | 1982 | $> 2^{21}$ | {0, 1} | 3221225471 |
| | | | | | | {3} | 3115801087 |
| | | | | | | {0, 3} | 3052757614 |

is normally distributed, and the test passes if $|Z| < 1.96$ given a 95% confidence level. Table 8.2 reports the values of Z for selected keystreams of the new SC (out of 64), including those whose statistic Z reaches the minimal and the maximal value. Clearly, as the maximal value of (15) remains acceptable, the test passes for all 64 examined keystreams.

*B. Diffusion test*

The diffusion (avalanche) effect refers to a behavior where a single flipped input bit leads to approximately half of the output bits being flipped at randomly distributed locations, making it statistically indistinguishable from random. In principle, SCs lack diffusion since each plaintext bit is mapped to a single ciphertext output bit. However, one can still verify if a single bit flip in an $s$-bit secret *initial value* (IV) of the cipher causes approximately half of the *keystream* bits being flipped at random [168], [180]. Recall that the initial values for the ciphers examined here consist of $s = m + n + 2h - 1$ bits. Consequently, to get $s$ statistics, the test proceeds as follows. Let H be an $k$-bin histogram, where $k$ is the keystream size. For every random IV $r$, it first produces $s$ sequences $r'$ by flipping all bits of $r$, one at a time. Every pair $(r, r')$ yields the keystreams $(d, d')$, and thus the value of $D = d \oplus d'$. Next, we increment $H(b)$ provided bit $b$ of D is set to 1. As can be seen, $H(b)$ counts the number of times the keystream bit $b$ has flipped in response to a single-bit change in the IV. After repeating the experiment $N \times s$ times, one can use the chi-square test with $k - 1$ degrees of freedom and the expected value of each bin being equal to N/2 to verify the hypothesis that histograms corresponding to successive bits of the IV represent uniformly distributed random variates. The second row of Table 8.2 only reports the worst cases, i.e., the largest values of $\chi^2$ among all $s$ statistics collected for selected keystreams of $k = 128$ bits each. Note that a critical value for this test equals $\chi^2 = 154.3$, provided that the significance level $\alpha = 0.05$.

*C. Correlation*

To validate whether a SC yields independent random keystreams, we measure a correlation between any pair of bits across N keystreams, collecting $k(k-1)/2$ correlation coefficients, $k = 128$. Clearly the correlation coefficient

$$\rho_{i,j} = N^{-1} \sum (x_i - 0.5)(x_j - 0.5) \tag{8.3}$$

between bits $x_i$ and $x_j$ should be close to 0 to confirm that there is no discernible relation between these two positions. This result should hold for all pairs of bits. Due to the large number of correlation coefficients, we only use their mean value S over all pairs $(i, j)$ and report the test statistic

$$Z = \sqrt{k(k-1)/2}(S - 0)/\sigma \tag{8.4}$$

that is expected to be normally distributed. The test passes provided $|Z| < 1.96$. Results are given in the third row of Table 8.2.

Table 8.2 Results of test from sections 8.4A, 8.4B, and 8.4C.

| Test | $L_1(1)$ | $L_1(2)$ | $L_1(3)$ | $L_2(1)$ | $L_2(2)$ | $L_2(3)$ | $L_3(1)$ | $L_3(2)$ | $L_3(3)$ | $L_4(1)$ | $L_4(2)$ | $L_4(3)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8.4A | 0.1092 | 0.215 | 0.4638 | 0.5061 | 0.2123 | 0.0957 | 0.2308 | 0.0981 | 0.8857 | 0.2102 | 0.3484 | 0.6535 |
| 8.4B | 85.891 | 91.624 | 88.271 | 83.461 | 81.624 | 79.958 | 92.03 | 82.799 | 84.297 | 83.937 | 85.825 | 91.048 |
| 8.4C | 0.5651 | 0.3347 | 1.7179 | 0.4045 | 0.3379 | 0.5977 | 0.4407 | 0.5781 | 0.6205 | 0.3397 | 1.8346 | 0.5526 |

## *D. Tuples test*

It examines distribution of $r$-bit overlapping patterns ($r$-tuples) in keystreams for values of $r$ ranging from 2 to 19 bits. To run every instance of this test, we form a $2^r$-bin histogram, and then, for every $r$-tuple occurring in a keystream, increment a bin corresponding to its value $k$. Let R and $B_k$ be the total number of $r$-tuples observed in a given keystream and the final value of bin $k$, respectively. The content of all bins can be statistically examined to see how closely they resemble a uniformly distributed random variate. This hypothesis is verified by the chi-square test using the statistic

$$\chi^2 = \sum_{k=0}^{2^r-1} \frac{(B_k - R/2^r)^2}{R/2^r} \tag{8.5}$$

which is approximately chi-square distributed with $2^r - 1$ degrees of freedom under the null hypothesis as in the diffusion test. Results of this test are gathered in Table 8.3 for selected

Table 8.3 Results of tuples test ($\chi^2$ values).

| $r$ | $L_1$ | $L_2$ | $L_3$ | $L_4$ | Critical values |
|---|---|---|---|---|---|
| 2 | 0.270635 | 0.450422 | 0.692174 | 1.08798 | 7.81 |
| 3 | 3.18169 | 0.729998 | 1.89032 | 1.84773 | 14.07 |
| 4 | 10.5418 | 1.8801 | 5.19466 | 13.2537 | 25.00 |
| 5 | 24.3318 | 9.37303 | 16.1516 | 28.6746 | 44.99 |
| 6 | 54.76 | 38.4467 | 52.8443 | 62.7262 | 82.53 |
| 7 | 115.457 | 99.9071 | 137.799 | 123.411 | 154.30 |
| 8 | 247.671 | 209.965 | 278.299 | 243.822 | 293.25 |
| 9 | 503.976 | 458.959 | 529.584 | 494.991 | 564.70 |
| 10 | 972.072 | 940.508 | 1053.45 | 1012.18 | 1098.52 |
| 11 | 1977.25 | 1925.03 | 2082.86 | 2039.82 | 2153.37 |
| 12 | 3957.5 | 3911.74 | 4107.22 | 4119.77 | 4244.99 |
| 13 | 8022.84 | 7943.5 | 8313.98 | 8285.45 | 8402.66 |
| 14 | 16116.7 | 16085.6 | 16606.8 | 16533.5 | 16681.9 |
| 15 | 32377.7 | 32451.4 | 33142.4 | 32785.7 | 33189.2 |
| 16 | 65098.7 | 64662.8 | 65945.1 | 65547.3 | 66131.6 |
| 17 | 130469 | 129764 | 131480 | 130504 | 131914 |
| 18 | 261427 | 260059 | 262383 | 261265 | 263335 |
| 19 | 522070 | 521419 | 525626 | 522565 | 525972 |

keystreams. The last column of the table provides critical values for successive values of *r*. Similar (passing) results were obtained for keystreams not reported here.

*E. NIST test suites*

The proposed cipher was examined using NIST Test Suite SP800-22 (see Chapter 7.2). Table 8.4 lists the results of all tests for chosen keystreams of the cipher. A number that follows the test name gives the total number of subtests. As can be verified, all examined keystreams passed all tests. The same applies to the remaining keystreams not reported here.

Table 8.4 Results for 1,000 1M-bit samples under NIST SP800-22 tests.

| | $L_1$ | | $L_2$ | | $L_3$ | | $L_4$ | |
|---|---|---|---|---|---|---|---|---|
| Test | Pass rate | PoP-value | Pass rate | PoP-value | Pass rate | PoP-value | Pass rate | PoP -value |
| Frequency - monobit (1) | 993/1000 | 0.440975 | 985/1000 | 0.678686 | 988/1000 | 0.422638 | 982/1000 | 0.032061 |
| Block frequency (1) | 992/1000 | 0.884671 | 989/1000 | 0.72987 | 992/1000 | 0.916599 | 993/1000 | 0.927677 |
| Cumulative sums (2) | 993/1000 | 0.668321 | 986/1000 | 0.498313 | 989/1000 | 0.424453 | 982/1000 | 0.215574 |
| | 993/1000 | 0.09372 | 985/1000 | 0.373625 | 992/1000 | 0.331408 | 982/1000 | 0.248014 |
| Runs (1) | 986/1000 | 0.649612 | 991/1000 | 0.363593 | 995/1000 | 0.234373 | 991/1000 | 0.394195 |
| Longest runs (1) | 992/1000 | 0.821937 | 985/1000 | 0.271619 | 992/1000 | 0.077131 | 990/1000 | 0.926487 |
| Matrix rank (1) | 988/1000 | 0.24549 | 990/1000 | 0.476911 | 990/1000 | 0.550347 | 990/1000 | 0.603841 |
| DFT - spectral (1) | 985/1000 | 0.542228 | 986/1000 | 0.401199 | 989/1000 | 0.968128 | 988/1000 | 0.572847 |
| Non-overlapping template (148) * | 982/1000 | 0.664168 | 982/1000 | 0.377007 | 982/1000 | 0.544254 | 982/1000 | 0.94008 |
| | 997/1000 | 0.194813 | 996/1000 | 0.94008 | 998/1000 | 0.314544 | 997/1000 | 0.526105 |
| Overlapping template (1) | 988/1000 | 0.72987 | 989/1000 | 0.607993 | 992/1000 | 0.063217 | 987/1000 | 0.55442 |
| Maurer's universal (1) | 988/1000 | 0.285427 | 991/1000 | 0.337688 | 991/1000 | 0.933472 | 988/1000 | 0.745908 |
| Approx. entropy (1) | 992/1000 | 0.674543 | 989/1000 | 0.607993 | 991/1000 | 0.514124 | 991/1000 | 0.947308 |
| Random excursions (8) | 599/606 | 0.712961 | 590/600 | 0.571477 | 626/635 | 0.158344 | 586/598 | 0.614191 |
| | 604/606 | 0.35319 | 594/600 | 0.888137 | 632/635 | 0.351772 | 592/598 | 0.445134 |
| | 600/606 | 0.108791 | 594/600 | 0.749884 | 627/635 | 0.182187 | 594/598 | 0.736521 |
| | 599/606 | 0.746572 | 592/600 | 0.200472 | 629/635 | 0.594234 | 596/598 | 0.70557 |
| | 603/606 | 0.019631 | 590/600 | 0.514124 | 628/635 | 0.452054 | 593/598 | 0.820813 |
| | 600/606 | 0.280306 | 594/600 | 0.81047 | 626/635 | 0.702896 | 594/598 | 0.805381 |
| | 601/606 | 0.024083 | 594/600 | 0.90242 | 623/635 | 0.660243 | 590/598 | 0.316001 |
| | 595/606 | 0.350485 | 591/600 | 0.924076 | 630/635 | 0.881013 | 589/598 | 0.084185 |
| Random excursions variant (18) * | 597/606 | 0.962959 | 590/600 | 0.175049 | 625/635 | 0.084192 | 589/598 | 0.534146 |
| | 604/606 | 0.378138 | 596/600 | 0.517442 | 635/635 | 0.132253 | 597/598 | 0.911413 |
| Serial (2) | 992/1000 | 0.174728 | 988/1000 | 0.205531 | 982/1000 | 0.53012 | 988/1000 | 0.106246 |
| | 994/1000 | 0.599693 | 987/1000 | 0.302657 | 987/1000 | 0.457825 | 992/1000 | 0.305599 |
| Linear complexity (1) | 980/1000 | 0.959347 | 993/1000 | 0.618385 | 990/1000 | 0.820143 | 990/1000 | 0.426272 |

*) Since this test consists of a large number of subtests, only the smallest and the largest values of the pass rate are reported herein altogether with the corresponding PoP-values.

The next tests belong to the NIST Test Suite SP800-90B [179]. A major part of the suit is comprised of tests that derive certain statistic $S_0$ based on the original test sequence, and then repeat the same test 10000 times for randomly permuted versions of the original sequence producing statistics $S_k$, $k = 1, \ldots, 10000$. If bits of the original sequence are independent and identically distributed (IID), then the statistics corresponding to permuted sequences will be similar to that of the original sequence. Every test employs counters $C_0$, $C_1$, and $C_2$ which are incremented, if $S_k > S_0$, $S_k = S_0$, or $S_k < S_0$, respectively. Finally, too high or too low counters indicate that tested data are non-IID. All tests use a cutoff value of 5 [179] such that if $C_0 + C_1 \leq 5$ or $C_1 + C_2 \leq 5$, then a test fails. Once it is known that none of these inequalities will be satisfied, the corresponding test terminates as passing. The experimental results for the same keystreams as before are presented in Table 8.5. It lists, for permutation tests, the counter values saved when a given test was terminated. Again, all keystreams passed those tests. They also passed four additional statistical tests [179], as shown in the table. The last row of Table 8.5 reports the min-entropy extracted from the keystreams by native functions of the SP800-90B suite.

Table 8.5 Results for 1,000 1M-bit samples under NIST SP800-90B IID tests.

| | $L_1$ | | | $L_2$ | | | $L_3$ | | | $L_4$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Test | $C_0$ | $C_1$ | $C_2$ | $C_0$ | $C_1$ | $C_2$ | $C_0$ | $C_1$ | $C_2$ | $C_0$ | $C_1$ | $C_2$ |
| Excursion | 6 | 0 | 6 | 8 | 0 | 6 | 6 | 0 | 6 | 6 | 0 | 48 |
| #Directional runs | 115 | 0 | 6 | 6 | 0 | 6 | 63 | 0 | 6 | 6 | 0 | 17 |
| Length of directional runs | 0 | 6 | 15 | 4 | 6 | 0 | 1 | 6 | 0 | 6 | 6 | 0 |
| #Increases and Decreases | 6 | 0 | 9 | 6 | 0 | 164 | 6 | 0 | 54 | 9 | 0 | 6 |
| #Runs wrt median | 14 | 0 | 6 | 6 | 0 | 7 | 7 | 0 | 6 | 18 | 0 | 6 |
| Length of runs wrt median | 4 | 2 | 19 | 11 | 2 | 4 | 3 | 3 | 8 | 265 | 6 | 0 |
| Average collision | 6 | 0 | 16 | 6 | 0 | 7 | 6 | 0 | 18 | 6 | 0 | 9 |
| Maximum collision | 16 | 5 | 1 | 18 | 2 | 4 | 3 | 3 | 5 | 4 | 2 | 7 |
| Compression | 6 | 0 | 317 | 6 | 0 | 6 | 6 | 0 | 626 | 12 | 0 | 6 |
| Periodicity-1 | 55 | 0 | 6 | 9 | 0 | 6 | 11 | 0 | 6 | 6 | 0 | 25 |
| Periodicity-2 | 6 | 0 | 21 | 22 | 0 | 6 | 12 | 0 | 6 | 68 | 0 | 6 |
| Periodicity-8 | 6 | 0 | 6 | 6 | 0 | 7 | 6 | 0 | 23 | 6 | 0 | 34 |
| Periodicity-16 | 6 | 0 | 11 | 6 | 0 | 33 | 67 | 0 | 6 | 31 | 0 | 6 |
| Periodicity-32 | 6 | 0 | 81 | 6 | 0 | 26 | 20 | 0 | 6 | 58 | 0 | 6 |
| Covariance-1 | 6 | 0 | 13 | 6 | 0 | 6 | 6 | 0 | 6 | 6 | 0 | 11 |
| Covariance-2 | 9 | 0 | 6 | 6 | 0 | 432 | 6 | 0 | 11 | 7 | 0 | 6 |
| Covariance-8 | 6 | 0 | 8 | 6 | 0 | 23 | 6 | 0 | 6 | 6 | 0 | 7 |
| Covariance-16 | 6 | 0 | 492 | 6 | 0 | 92 | 86 | 0 | 6 | 6 | 0 | 6 |
| Covariance-32 | 6 | 0 | 19 | 258 | 0 | 6 | 40 | 0 | 6 | 6 | 0 | 87 |
| Chi-square independence | *p*-value = 0.716681 | | | *p*-value = 0.861359 | | | *p*-value = 0.120963 | | | *p*-value = 0.474993 | | |
| Chi-square goodness of fit | *p*-value = 0.154031 | | | *p*-value = 0.262234 | | | *p*-value = 0.864069 | | | *p*-value = 0.712502 | | |
| Length of the longest repeated substring | Passed | | | Passed | | | Passed | | | Passed | | |
| Restart | Passed | | | Passed | | | Passed | | | Passed | | |
| Min. entropy per byte | 7.987918 | | | 7.987513 | | | 7.988981 | | | 7.988605 | | |

*F. AIS test suite*

The last group of tests come from the AIS-31 Test Suite [91] that consists of 9 tests, commonly referred to as T0–T8. T0 requires at least $2^{16} \cdot 48$ bits to be conducted. Tests from T1 to T5 are repeated 257 times on consecutive 20000-bit long parts of a tested sequence. T0 through T5 aim at evaluating the randomness of the tested sequence. Tests T6–T8 works with a 7200000-bit sequence. Their purpose is to find the entropy level of the tested sequence. The pass rates corresponding to T1–T5 are reported in Table 8.6. Furthermore, the lower part of the same table gives the test statistics obtained for the examined keystreams (the passing criteria for each test can be found in the second column of the table). As can be seen, the keystreams pass all statistical tests of AIS-31. In particular, the pass rate for T0–T5 is 100%.

Table 8.6 Results for binary sequences under AIS-31 tests.

| Test | | $L_1$ | $L_2$ | $L_3$ | $L_4$ |
|---|---|---|---|---|---|
| | | Pass rate | | | |
| T0 | Disjointness | Passed | Passed | Passed | Passed |
| T1 | Monobit | 257/257 | 257/257 | 257/257 | 257/257 |
| T2 | Poker | 257/257 | 257/257 | 257/257 | 257/257 |
| T3 | Run | 257/257 | 257/257 | 257/257 | 257/257 |
| T4 | Long run | 257/257 | 257/257 | 257/257 | 257/257 |
| T5 | Autocorrelation | 257/257 | 257/257 | 257/257 | 257/257 |
| | | Test values (Statistics – S) | | | |
| T6 - a | Uniform distr. ( $S < 0.025$) | 0.00105 | 0.00086 | 0.00166 | 0.00103 |
| T6 - b | Uniform distr. ($S < 0.020$) | 0.00088 | 0.00010 | 0.00189 | 0.00254 |
| T7 - a | Comparative multinomial, width = 3 ($S < 15.13$) | 8.65930 | 0.32258 | 3.61251 | 1.77609 |
| | | 0.13778 | 1.10450 | 1.80001 | 0.35379 |
| T7 - b | Comparative multinomial, width = 4 ($S < 15.13$) | 1.22018 | 0.05408 | 3.42792 | 1.29032 |
| | | 0.46818 | 0.08450 | 0.64082 | 0.29282 |
| | | 0.03872 | 0.86528 | 0.29282 | 3.74980 |
| | | 9.80003 | 0.34322 | 0.55112 | 0.07688 |
| T8 | Entropy ($S > 7.976$) | 8.00457 | 8.00215 | 7.99972 | 7.99681 |

*G. Hardware footprint*

As shown in Chapter 8.1, the proposed cipher requires ($m + n + 2h – 1$) FFs, where $m$ and $n$ are sizes of NLFSRs, while $h$ is the PHRG size (additional $h – 1$ FFs form MR). Logic gates are also needed to implement all linear and nonlinear functions, data injectors, reset logic, etc. Their quantity depends primarily on feedback functions of involved shift registers. Table 8.7 reports the silicon real estate taken up by the examined instances of the cipher in terms of GE, where one GE is the area occupied by a 2-input NAND gate, as mentioned in Chapter 7.2. The presented numbers were obtained with a commercial synthesis tool working with a 7nm CMOS standard cell library. In addition to clock tree synthesis, the deployed design flow was comprised of placement and route with intermediate timing optimizations. Table 8.7 reports the number of combinational and sequential cells, the number of buffers/inverters,

Table 8.7 Hardware footprint – equivalent 2-input NAND gates.

| | Nets | Combina-tional cells | Sequential cells | Buffers inverters | References | Combina-tional area | Buf/inv area | Sequential area | Total area | Speed [GHz] | Power [μW] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $L_1$ | 1825 | 1825 | 105 | 138 | 23 | 2078 | 104 | 464 | 2542 | 6.5 | 468.48 |
| $L_2$ | 1966 | 1226 | 131 | 164 | 26 | 2204 | 123 | 581 | 2784 | 6.2 | 507.58 |
| $L_3$ | 2235 | 1575 | 185 | 219 | 27 | 2429 | 165 | 824 | 3252 | 5.9 | 535.04 |
| $L_4$ | 2632 | 2632 | 264 | 264 | 26 | 2745 | 236 | 1179 | 3924 | 5.8 | 695.12 |

and the number of references, i.e., library components used in building larger blocks. The corresponding GE numbers with respect to the aforementioned cells are listed in the right part of the same table. Finally, the last two columns of the table provide a maximal operating frequency of the cipher after assessing its critical paths and results of statistical power analysis based on the circuits' switching activity at 400 MHz.

## 8.5 Resilience against attacks

SCs, like other cryptographic primitives, can be exposed to malicious activities. This chapter briefly summarizes arguments applicable to the resilience of the proposed SCs against several attacks and cryptanalysis techniques. Typically, attacks mounted on SCs are aimed at compromising secret keystreams. Therefore, it is crucial to note that initial values are unique per each keystream generation. By having the initial values selected this way, the proposed cipher increases its resilience against most of the attacks discussed in the following paragraphs.

*Algebraic attacks.* Here the attacker tries to solve a system of nonlinear equations, typically by replacing all products of variables with a single variable and solving the resultant system of linear equations. Consequently, a SC is regarded resilient if its nonlinear functions have a high degree. Though the cipher consists of a linear HRG, selection of its feedback function is carried out by virtue of a secret key. To mount an algebraic attack, one must first determine the content of the selector, which is represented by nonlinear equations. Moreover, data from PHRG are used to control the outputs of (nonlinear) multiplexers, which are powered by two different NLFSRs. As can be seen, complex nonlinear equations are deployed at each stage of the keystream generation process. These properties can make an algebraic attack on the cipher infeasible within a reasonable period of time.

What may enable a *correlation attack* is a statistically biased encryption implied by certain internal state variables used as inputs. Hence, guessing such input bits would most likely impact the actual output bits. With the help of statistical tools, the adversary can recover these internal values. In addition to results of Chapter 8.4C, nonlinear components of the cipher (such as a nonlinear selection of PHRG feedback functions or both NLFSRs) make it resistant to correlation-based attacks.

*Time-memory trade-off attacks* can be successful provided a cipher state space is relatively small. During a preprocessing phase, the attacker generates many initial states and the corresponding keystream prefixes that can be compared against a captured keystream. If there is a match, the initial state is found and a secret key might be retrieved. As the state space of the cipher can be freely enlarged (one of its instances examined in this chapter has more than $10^{57}$ states), it reduces or even precludes any chance of finding, in a time-acceptable manner, a keystream prefix identical with one of those precomputed earlier. As a result this attack is not feasible.

For *divide and conquer attacks* to work a cipher must be broken down into several rounds of computationally simpler functions. As shown earlier, the proposed cipher makes all input bits spread uniformly across all its registers. As a result, having a given bit (variable) appearing in output formulas is equally likely for all keystream bits. It is, therefore, virtually impossible to divide the attack into functions of a low complexity, where each one of them would be able to recover mutually exclusive bits from the secret key.

*Differential attacks* exploit nonuniform propagation of differences within blocks of a given cryptographic primitive. Even flipping a single initial bit and observing the resultant keystream may reveal its value if the secret key bits are nonuniformly distributed over the keystream bits. Fortunately, initialization cycles allow uniform and random diffusion of secret key bits. In particular, toggling rates for each input bit, observed at each output bit, do not leak any systematic information regarding the secret key (see Table 8.2).

*Fault attacks* assume that an adversary can inject bit-flipping faults by either varying a clock, voltage, temperature, or by using a laser beam. These methods exploit various circuit characteristics and typically have low spatial precision [73]. Although fault injection may expose some input-output relations as far as a linear PHRG is concerned (provided an exact location of a fault can be determined), the same hardly applies to both NLFSRs as fault propagation within nonlinear structures is much more difficult to predict and to take advantage of.

To cope with *chosen-IV attacks*, initial states of a cipher for any two chosen IVs should be statistically (and algebraically) unrelated. As shown in the previous chapter, even two IVs that differ on a single bit position yield keystreams where each of the output bits changes with a 50% probability (a Hamming weight of their difference is a random value). The same applies to IVs with multiple differences. It allows the cipher to resist this type of attacks.

To launch a *guess-and-determine attack* the adversary takes advantage of the fact that certain internal bits of the cipher can be deduced from other internal bits (a guess basis). Thus, given a guess basis of a minimal size, one derives the remaining bits, computes the output and evaluates its consistency against the corresponding keystream bits that were eavesdropped. A high diffusion level of the cipher makes this technique impractical as it requires a large guess basis.

*Malleability attacks.* A cipher is “malleable” if one can transform a ciphertext into another ciphertext decrypting to a desired plaintext. Virtually all SCs are vulnerable to this attack as they produce a keystream independently of a plaintext. Consequently, another cryptographic technique is required, for instance, based on a message authentication code, to protect data. As shown earlier, the proposed cipher is a part of a root-of-trust ecosystem whose other components define the final security of ICs. Furthermore, it appears that mounting malleability attacks is not that simple [14]. For example, an adversary must know exactly where and when the target test data is transmitted to be able to tamper with it. Bit-flipping at wrong positions is meaningless and will most likely result in a corrupted transmission.

*Side-channel attacks* have already been described in Chapter 7.3. As the cipher is primarily deployed as a part of an embedded hardware root of trust, its logic remains a negligible fraction of the entire chip which effectively acts as a source of additive noise distorting any measurable physical characteristic of either the cipher or the root of trust as a whole, and thus reducing the amount of information in a potential side-channel leakage.

# 9. Lightweight true random number generator

This chapter presents a lightweight design of a nonce generator for the root of trust applications [141]. It appears that the scheme is capable of fulfilling requirements for an efficient and secure TRNG. The presented ring-generator-based TRNGs pass a variety of well-known statistical tests and feature a low and scalable hardware footprint which makes them suitable for both ASIC and FPGA implementations of embedded systems that form a base of contemporary security solutions.

## 9.1 Motivation

As mentioned in Chapter 5.4, the root of trust security relies on its authentication protocol. What is characteristic for the challenge-response procedure, is its initialization phase that employs unique, random token (*nonce*). The nonce is typically produced by an IC-integrated TRNG that should yield different combinations of 0s and 1s every time it is activated. In light of the above, it is clear that TRNGs have become key hardware security primitives capable of producing random sequences by harvesting the randomness present in physical processes such as the thermal instability and noise [21], [22], [88], [111], [133], metastability [62], [76], [92], [108], [176], [177], [190], edge racing in digital designs [203], chaotic behavior of cellular automata [63], [82], [104], [132], power supply variations [175], stochastic nature of magnetic tunnel junction [183], quantum effects [169], or phase jitters in ring oscillators (ROs) [162]. The latter approach has gained noticeable popularity because it provides a simple yet effective method to build random number generators just by chaining an odd number of inverters into a ring structure. As a result, a wide range of solutions using this principle and its derivatives have been proposed in the contemporary technical literature and industrial practice. For the sake of illustration, let us recall a few exemplary solutions.

The most straightforward mechanism to extract randomness from a jitter is to sample the output of a RO using the output signal of another RO (Fig. 9.1a). Such coupled oscillators are presented in [3], [22], [51], [155], [200]. In [51], two ROs are coupled by a non-linear circuit, whereas in [200] the first RO feeds a programmable delay chain that is sampled by a bit extractor driven by the other RO. If periods of both signals are very close to each other,

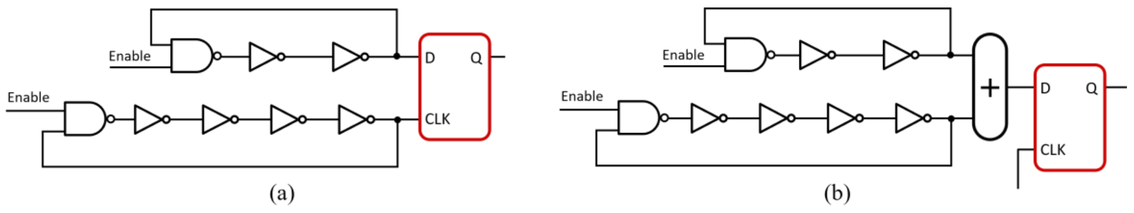


Figure 9.1 Conventional RO-based TRNG architectures: (a) sampling RO by another RO, (b) combining ROs to form a single sequence.

they form a basis for coherent sampling [95], [131], [202]. Alternatively, one can combine the output signals of several ROs by means of XOR trees (Fig. 9.1b), as shown in [4], [100], [160], [171], [198], or [199]. In particular, the work of [169] provides a thorough mathematical treatment of an approach where combined jitter signals form a source of entropy. A low power scheme where two identical ROs enable, through an XOR gate, a third RO clocking a counter is described in [26]. A reconfigurable TRNG based on transient effect ring oscillators with two different sampling methods is introduced in [1]. In [37], four ROs drive associated LFRSs whose outputs are sampled through a multiplexer driven by yet another RO. ROs with a multi-stage feedback structure can be XOR-ed to produce true random numbers, as detailed in [41]. Somehow different approach is discussed in [50], [52], [69], [127]; it implements an RO by replacing a simple circular feedback with a more complex network comprising XOR gates in a way corresponding to conventional Fibonacci or Galois LFSRs. Here, inverters replace memory elements. To enhance the performance of RO-based TRNG, one can also deploy Muller C-gates instead of inverters. These elements are then interleaved to form an asynchronous pipeline that is capable of propagating several simultaneous voltage events sampled by an XOR tree [33], [34]. Finally, different RO-based TRNGs are compared in [134] to demonstrate how they are amenable to FPGA-based implementations. One may also find interesting details in other relevant papers such as [18], [23], [31], [49], [107], [152], [176], and [182].

Besides the schemes recalled above, there are other techniques deployed to produce truly random sequences of bits. Those schemes include the use of chaotic maps [13] with von Neumann correction algorithm [185], sampling a jitter in a phase-locked loop circuitry [64], extracting a design fingerprint during the power-up of SRAMs [81], or detecting a beat frequency in FPGAs [86].

In addition to its unpredictability, a modern TRNG design is expected to be easily synthesizable by using exclusively digital components [171], i.e., no amplifiers or other analog devices are allowed. Furthermore, additional post-processing steps and the corresponding circuitry to adjust the sampling frequency or to increase the per-bit entropy [160] should be avoided. Consequently, this chapter proposes a high-performance device that may assume the role of a lightweight all-digital TRNG. Although it was originally designed as a hardware generator of one-time challenges produced for the sake of IC authentication protocols, many tests have confirmed that it can be considered as a reliable source of truly random numbers used in a variety of cryptographic or security-related applications. The proposed design rests on a ring generator architecture [125] harvesting a source of entropy implemented by a conventional free-running ring oscillator, and further processing the captured data due to its feedback network.

## 9.2 General architecture

Fig. 9.2 illustrates the proposed scheme. Its major part is a ring generator [121], [125], as shown in Fig. 9.2 for the polynomial $h(x) = x^{32} + x^{27} + x^{21} + x^{16} + x^{10} + x^5 + 1$. Recall that ring generators, in a vivid contrast to the corresponding Fibonacci or Galois LFSRs, offer a certain degree of flexibility in forming its structure. Every resultant device will feature a different state trajectory; all of them, however, will remain maximum-length finite state machines producing the same *m*-sequence, just differently phase-shifted in each case [121].

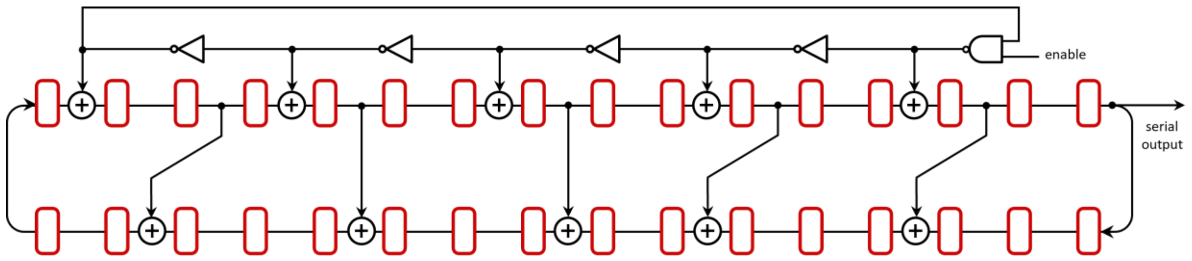


Figure 9.2 Ring-generator-based true random number generator with the characteristic polynomial $h(x) = x^{32} + x^{27} + x^{21} + x^{16} + x^{10} + x^5 + 1$.

Since every ring generator is a planar, high-speed circuit that features reduced internal fan-outs and minimal delays on critical paths, it causes no frequency degradation and lets designers minimize routing complexity, optimize wire sizing, and make the overall layout as compact as possible [125]. Clearly, a synchronous ring generator has deterministic behavior which renders the generated values as predictable as any other LFSR-produced pseudorandom sequences (vulnerable to statistical attacks). To make the ring generator suitable for TRNG-based applications, it is enhanced by adding an entropy source implemented as a free-running *m*-stage gated ring oscillator, as shown in Fig. 9.2, where the oscillator is made of a single NAND gate and four inverters chained into a ring. The oscillator internal signals, sampled at the outputs of selected inverters, are subsequently injected into the ring generator via XOR gates placed in the front of its "upper level" flip-flops. Consequently, the ring generator acts as a special form of a bit extractor processing data collected at several RO stages. Similar techniques, i.e., the entropy extraction from a single transition event, are also discussed in [159], [200], and [202], though the latter two works deploy an extra RO-driven tapped delay chain placed outside of a ring oscillator. Furthermore, since the clocking of the ring generator is inherently asynchronous to the state of a ring oscillator, some clock samples may stress the metastability region of the ring generator flip-flops (due to setup and hold time violations), thereby producing an additional and desired uncertainty (entropy) or randomness. Note that the new design does not require any additional zero detector similar to that of [104]. It is also worth recalling that upon every successful authentication of a given IC, a challenge-response pair can be potentially revealed to the adversary and thus cannot be used again. Fortunately, the inherent feature of the proposed TRNG is its ability to produce a different random value virtually every time it is invoked or reset.

It is worth noting that basic theoretical foundations which the new TRNG depends on remain similar to those presented elsewhere. Indeed, as an instance of a linear finite state machine sampling a ring oscillator, a modus operandi of the new scheme resembles principles analyzed mathematically in several earlier publications. In particular, the urn model of [171] is used to analyze random data sampling by virtue of a combinatorial approach. Moreover, a detailed entropy model of a ring of identical units acting as a source of chaotic behavior has been proposed and meticulously discussed in [63] and [104].

## 9.3 First validation steps

To validate the proposed TRNG scheme, its instances of varying sizes (see Table 9.1) were implemented on a single Xilinx Artix-7 FPGA chip using the Digilent Arty Z7-200 board with a port that facilitates data collection. It is important to note that all results presented in this chapter (with the exception of the entropy estimation) come from the same FPGA platform and were obtained for $n$-bit numbers $b_{n-1}b_{n-2} \ldots b_1b_0$, where $n$ is the TRNG size, $n \in [32, 256]$. In order to mimic the challenge generator behavior, every circuit was powered up 100,000 times and the resultant n-bit values were scanned out after $2^{11}$ clock cycles. This process yielded 100,000 n-bit numbers for further examination. Fig. 9.3 plots the distribution of 0s and 1s in 64-bit numbers produced by the new TRNG using the primitive polynomial $G_{64} = x^{64} + x^{52} + x^{39} + x^{26} + x^{14} + x^{7} + 1$ and working with a ring oscillator of 7 inverters. Fig. 9.3 illustrates the first (successive) 768 random samples (nonces) obtained this way. They are arranged into four columns such that the first 192 samples are displayed in the first column, the next 192 samples are placed in the second column, etc.

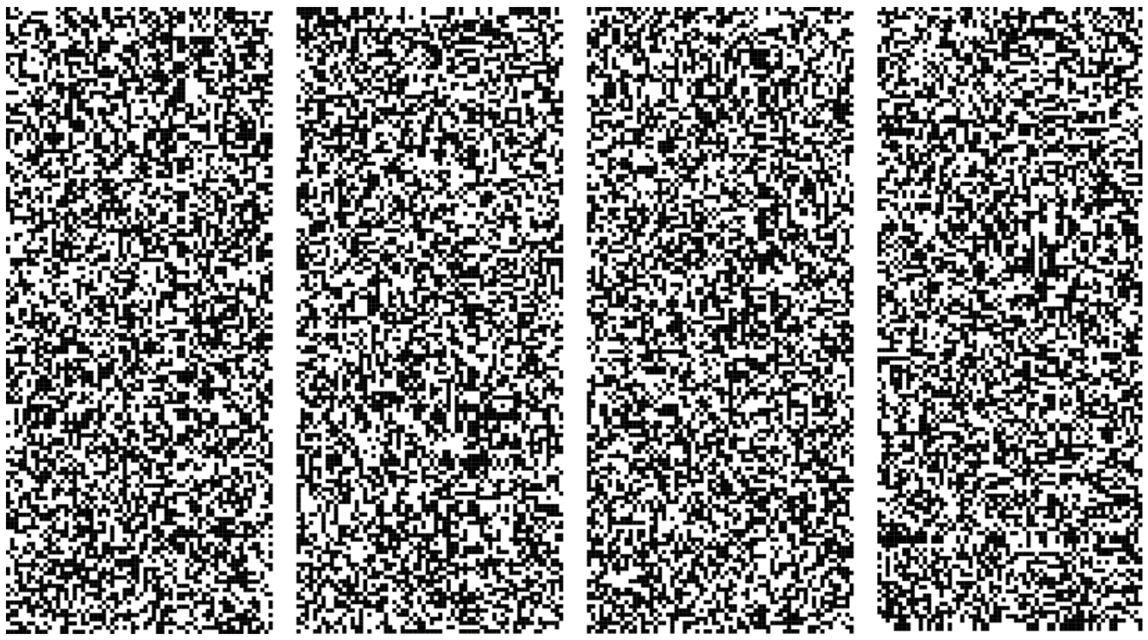


Figure 9.3 Distribution of 64-bit random values after power-up and $2^{11}$ cycles.

An ideal TRNG has to yield independent random binary combinations as otherwise its behavior could be easily anticipated. To validate this feature, one can measure a correlation between any pair of bits across all sampled random outputs, effectively collecting $n(n-1)/2$ correlation coefficients. Clearly, given $s$ successive samples (in this case $s$ = 100,000), the correlation coefficient

$$\rho_{i,k} = s^{-1} \sum (b_i - 0.5)(b_k - 0.5) \tag{9.1}$$

between bits $b_i$ and $b_k$ should be close to 0 to confirm that there is no strong, discernible, and systematic relation between these two positions. Such a result should hold for all pairs of bits. The test covered random numbers produced by target $n$-bit TRNGs, taking 100,000 samples in each case. Consider, for the sake of illustration, a 64-bit TRNG (its polynomial is given in Table 9.1). It turns out that the mean value of the correlation over all $(64 \cdot 63)/2$ = 2,016 pairs of bits is about $\mu \approx -2.731\text{e} \cdot 10^{-5}$, whereas its standard deviation is $\sigma \approx 0.003509$. Moreover, the minimal value is equal to 0.0 for $\rho_{4,2}$, the maximal value (absolute) is equal to $\rho_{21,12}$ = 0.002830, and |$z$-statistic| = 0.349469 < 1.96, for a 95% confidence level. In fact, none of the recorded coefficients was significantly different from 0 in comparison with the normal distribution, thus indicating that the produced samples do not exhibit observable correlation between any pair of their bits. Similar results were obtained for other TRNGs. Their characteristics, i.e., a primitive polynomial used to setup a ring generator and the number of injectors provided by an 11-inverter ring oscillator, are listed in Table 9.1, whereas the correlation results are gathered in the *left* part of Table 9.2.

Another simple empirical test is aimed at checking whether the logic value of 1 occurs on every bit position roughly half of the time (50%). This hypothesis can be verified by, for

Table 9.1 TRNG characteristics.

| ID | Polynomial | #IN |
|---|---:|---|
| $G_{32}$ | 32 25 17 6 0 | 6 |
| $G_{48}$ | 48 35 22 10 0 | 5 |
| $G_{64}$ | 64 56 49 40 31 24 16 8 0 | 4 |
| $G_{80}$ | 80 62 37 15 0 | 5 |
| $G_{96}$ | 96 73 43 18 0 | 5 |
| $G_{112}$ | 112 101 92 83 73 62 52 42 32 22 0 | 5 |
| $G_{128}$ | 128 95 66 29 0 | 7 |
| $G_{144}$ | 144 113 79 37 0 | 7 |
| $G_{160}$ | 160 144 125 101 79 37 0 | 5 |
| $G_{176}$ | 176 169 162 153 144 136 128 120 112 104 96 88 80 72 64 56 48 40 0 | 6 |
| $G_{192}$ | 192 169 144 96 70 52 39 26 13 0 | 8 |
| $G_{208}$ | 208 172 139 105 70 35 0 | 8 |
| $G_{224}$ | 224 206 182 155 131 105 80 54 36 18 0 | 7 |
| $G_{240}$ | 240 221 200 181 162 141 120 100 80 60 40 20 0 | 8 |
| $G_{256}$ | 256 212 169 127 86 42 0 | 6 |

Table 9.2 Correlation and min-entropy.

| | Correlation | | | Min-entropy | |
|---|---|---|---|---|---|
| | Mean | Std. | z-statistic | FPGA | Simulation |
| $G_{32}$ | -2.52E-05 | 0.001785 | -0.314596 | 0.99282 | 0.980947 |
| $G_{48}$ | -1.30E-06 | 0.002676 | -0.016295 | 0.993773 | 0.987687 |
| $G_{64}$ | -2.73E-05 | 0.003509 | -0.349469 | 0.996669 | 0.981135 |
| $G_{80}$ | -6.71E-06 | 0.004447 | -0.084801 | 0.995552 | 0.981528 |
| $G_{96}$ | 1.95E-05 | 0.005342 | 0.246434 | 0.993806 | 0.984408 |
| $G_{112}$ | 1.09E-05 | 0.006234 | 0.138298 | 0.991189 | 0.986258 |
| $G_{128}$ | -3.45E-06 | 0.007168 | -0.043407 | 0.991438 | 0.985751 |
| $G_{144}$ | -2.42E-06 | 0.008018 | -0.030578 | 0.991112 | 0.982384 |
| $G_{160}$ | 4.90E-06 | 0.008909 | 0.06198 | 0.997344 | 0.987837 |
| $G_{176}$ | -1.83E-06 | 0.009844 | -0.023005 | 0.990098 | 0.981029 |
| $G_{192}$ | 2.21E-06 | 0.010724 | 0.027936 | 0.990169 | 0.98616 |
| $G_{208}$ | -1.93E-06 | 0.011658 | -0.024337 | 0.991306 | 0.981105 |
| $G_{224}$ | 2.45E-06 | 0.012586 | 0.030701 | 0.991235 | 0.985696 |
| $G_{240}$ | 3.14E-06 | 0.0135 | 0.039426 | 0.991911 | 0.983453 |
| $G_{256}$ | -9.61E-07 | 0.01435 | -0.012093 | 0.993663 | 0.98437 |

instance, the chi-square test. Thus, the histogram of 1s observed on successive bits of the 64-bit-TRNG-produced numbers is shown in Fig. 9.4. Here, every bin displays the percentage of 1s observed on every bit of 100,000 samples that were collected. Similarly, the number of *n*-bit sequences with the Hamming weight *k* should be binomially distributed, as illustrated in Fig. 9.5. Again, the goodness-of-fit hypothesis test can be used to validate this observation. Both tests were employed to successfully validate the ring-generator-based TRNGs listed in Table 9.1.

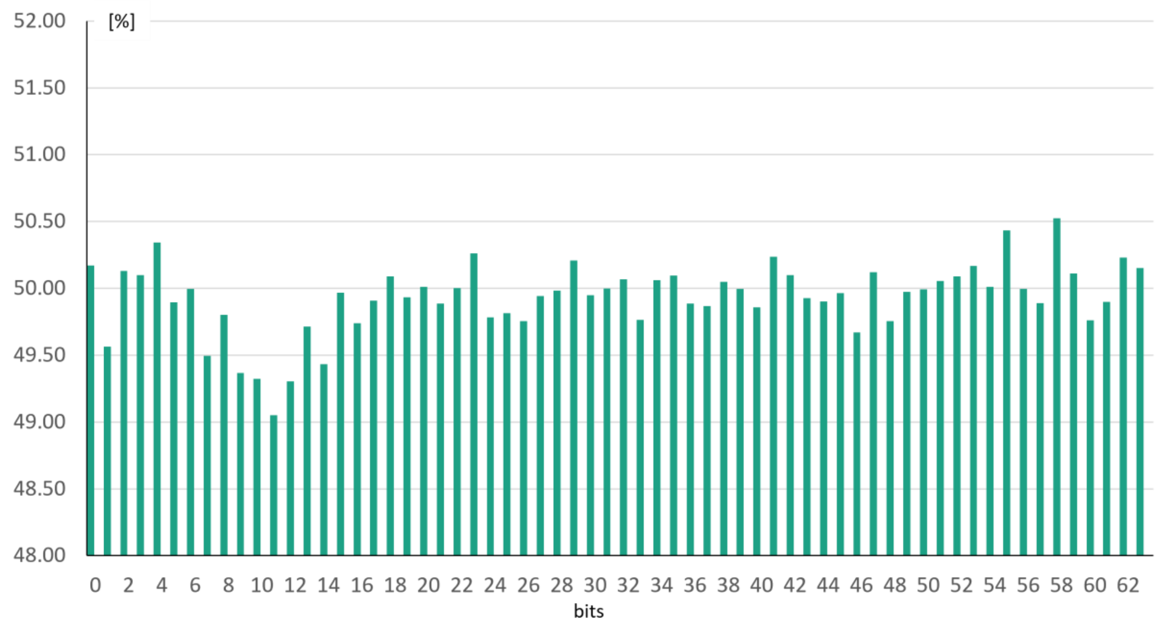


Figure 9.4 The fraction of 1s on successive locations in 64-bit random samples.

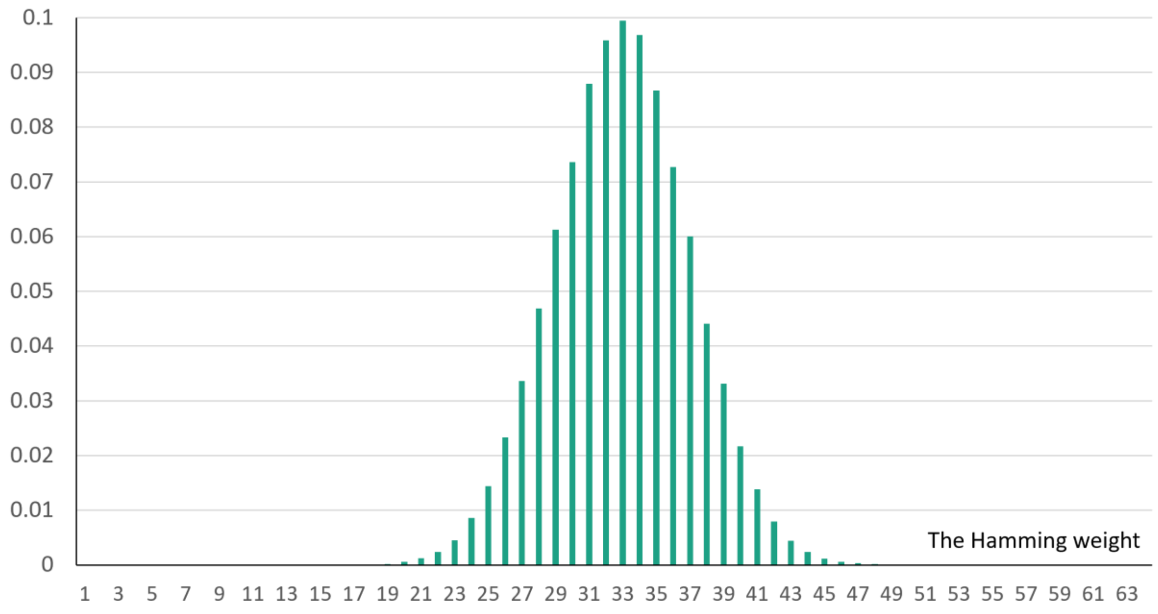


Figure 9.5 Distribution of 64-bit samples wrt to their Hamming weights.

The ability of a TRNG to deliver secure random values requires sufficient entropy on its outputs. In order to estimate the amount of entropy produced per bit by the proposed scheme, the min-entropy estimate based on the collision count was used[74]. It measures the mean number of samples to the first repeated value in a binary sequence. The method assesses the probability of the most-likely output value based on the collision times. Consequently, it produces a low entropy estimate for sources that have a considerable bias toward a particular value, while yielding a higher entropy estimate for a longer mean time to collision [179]. In the experiments, serial bit sequences observed on a selected output of the TRNG (see Fig. 9.2) and comprising $10^9$ bits were harvested both from the FPGA-based implementations as well as from a simulation model of the proposed scheme. In the latter case, event-driven simulation experiments were run by assuming that the mean delay of individual gates of a ring oscillator is set to 280 ps, while their random Gaussian jitter has the mean value of 0 and the standard deviation of 30 ps, similarly to a simulation setup described in [18]. The sampling clock driving the ring generator had 2.5 ns time period (or 400 MHz frequency). Its flip-flops had 20 ps setup time. Results of both types of experiments are presented in the *right* part of Table 9.1. As can be seen, the lower bound of entropy is above 0.99 per bit based on data received from the FPGA setups. The min-entropy recorded and determined by means of the simulation model was measured to be above 0.98 per bit. It is also worth noting that in all reported experiments the Shannon entropy given by

$$\mathrm{H} = -p_1 \cdot \log_2(p_1) - (1 - p_1) \cdot \log_2(1 - p_1)$$

was always greater than 0.999999, which is higher than 0.997 per bit requested by AIS-31 [91] ($p_1$ is the probability of having 1 in the examined binary sequence).

## 9.4 More experimental results

This chapter thoroughly evaluates the performance of the proposed TRNG design using two statistical test suites [12], [179] from the NIST and another test set (AIS-31) provided by the German IT security certification authority (BSI) [91]. Their brief descriptions can be found in Chapter 7.2 and Chapter 8.4. Binary sequences subjected to various tests were serially collected from FPGA implementations on a single output of a ring generator (see Fig. 9.2). After an initialization period of 2*s* cycles, where *s* is the ring generator size, TRNG was clocked $10^9$ times, thereby producing 1,000 consecutive sequences, each comprising $10^6$ bits.

Detailed results of applying the NIST SP800-22 [12] tests to binary sequences produced by four selected TRNGs are listed in Table 9.3. A number in brackets that follows the test name gives the total number of subtests a given test consists of. As can be seen, all TRNGs pass all tests. The same applies to the remaining TRNGs of Table 9.1; their detailed results are not shown here because of space constraints.

Table 9.3 Results for 1,000 1M-bit samples under NIST SP800-22 tests.

| | $G_{48}$ | | $G_{64}$ | | $G_{128}$ | | $G_{256}$ | |
|---|---|---|---|---|---|---|---|---|
| Test | Pass rate | PoP-value | Pass rate | PoP-value | Pass rate | PoP-value | Pass rate | PoP -value |
| Frequency - monobit (1) | 989/1000 | 0.985339 | 990/1000 | 0.177628 | 988/1000 | 0.622546 | 988/1000 | 0.622546 |
| Block frequency (1) | 988/1000 | 0.930026 | 987/1000 | 0.735908 | 993/1000 | 0.922855 | 993/1000 | 0.922855 |
| Cumulative sums (2) | 987/1000 | 0.420827 | 992/1000 | 0.917870 | 989/1000 | 0.350485 | 989/1000 | 0.350485 |
| | 990/1000 | 0.500279 | 992/1000 | 0.212184 | 990/1000 | 0.347257 | 990/1000 | 0.347257 |
| Runs (1) | 991/1000 | 0.267573 | 991/1000 | 0.849708 | 992/1000 | 0.998971 | 992/1000 | 0.998971 |
| Longest runs (1) | 988/1000 | 0.890582 | 993/1000 | 0.805569 | 988/1000 | 0.678686 | 988/1000 | 0.678686 |
| Matrix rank (1) | 987/1000 | 0.846338 | 989/1000 | 0.992381 | 992/1000 | 0.211064 | 992/1000 | 0.211064 |
| DFT - spectral (1) | 987/1000 | 0.893482 | 985/1000 | 0.034942 | 988/1000 | 0.821937 | 988/1000 | 0.821937 |
| Non-overlapping template (148) * | 980/1000 | 0.579021 | 982/1000 | 0.325206 | 984/1000 | 0.605916 | 984/1000 | 0.605916 |
| | 996/1000 | 0.715679 | 997/1000 | 0.289667 | 997/1000 | 0.045088 | 997/1000 | 0.045088 |
| Overlapping template (1) | 994/1000 | 0.597620 | 982/1000 | 0.388990 | 990/1000 | 0.397688 | 990/1000 | 0.397688 |
| Maurer’s universal (1) | 993/1000 | 0.024855 | 994/1000 | 0.562591 | 988/1000 | 0.063615 | 988/1000 | 0.063615 |
| Approx. entropy (1) | 992/1000 | 0.142062 | 990/1000 | 0.404728 | 993/1000 | 0.289667 | 993/1000 | 0.289667 |
| Random excursions (8) | 615/619 | 0.681400 | 604/611 | 0.613238 | 606/614 | 0.841394 | 606/614 | 0.841394 |
| | 612/619 | 0.731687 | 603/611 | 0.038964 | 610/614 | 0.096959 | 610/614 | 0.096959 |
| | 613/619 | 0.532495 | 608/611 | 0.691554 | 611/614 | 0.733338 | 611/614 | 0.733338 |
| | 613/619 | 0.243497 | 604/611 | 0.864547 | 608/614 | 0.158932 | 608/614 | 0.158932 |
| | 614/619 | 0.848358 | 604/611 | 0.069317 | 603/614 | 0.026771 | 603/614 | 0.026771 |
| | 617/619 | 0.914620 | 600/611 | 0.206917 | 605/614 | 0.160394 | 605/614 | 0.160394 |
| | 613/619 | 0.176419 | 607/611 | 0.080680 | 609/614 | 0.83 | 609/614 | 0.830000 |
| | 612/619 | 0.239427 | 604/611 | 0.099367 | 606/614 | 0.073305 | 606/614 | 0.073305 |
| Random excursions variant (18) * | 610/619 | 0.019058 | 601/611 | 0.103329 | 605/614 | 0.614942 | 605/614 | 0.614942 |
| | 618/619 | 0.260305 | 609/611 | 0.341275 | 611/614 | 0.733338 | 611/614 | 0.048460 |
| Serial (2) | 993/1000 | 0.073417 | 989/1000 | 0.699313 | 986/1000 | 0.045088 | 986/1000 | 0.045088 |
| | 997/1000 | 0.181557 | 989/1000 | 0.480771 | 988/1000 | 0.274341 | 988/1000 | 0.274341 |
| Linear complexity (1) | 985/1000 | 0.757790 | 991/1000 | 0.422638 | 993/1000 | 0.741918 | 993/1000 | 0.741918 |

*) Since this test consists of a large number of subtests, only the smallest and the largest values of the pass rate are reported herein altogether with the corresponding PoP-values.

Table 9.4 Results for 1,000 1M-bit samples under NIST SP800-90B IID tests.

| Test | $G_{48}$ | | | $G_{64}$ | | | $G_{128}$ | | | $G_{256}$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $C_0$ | $C_1$ | $C_2$ | $C_0$ | $C_1$ | $C_2$ | $C_0$ | $C_1$ | $C_2$ | $C_0$ | $C_1$ | $C_2$ |
| Excursion | 32 | 0 | 6 | 13 | 0 | 6 | 6 | 0 | 10 | 6 | 0 | 11 |
| #Directional runs | 6 | 0 | 28 | 6 | 0 | 44 | 10 | 0 | 6 | 8 | 0 | 6 |
| Length of directional runs | 4 | 6 | 0 | 2 | 5 | 1 | 1 | 5 | 16 | 8 | 6 | 0 |
| #Increases and Decreases | 6 | 0 | 21 | 6 | 0 | 10 | 9 | 0 | 6 | 6 | 0 | 104 |
| #Runs wrt median | 11 | 0 | 6 | 29 | 0 | 6 | 6 | 0 | 26 | 15 | 0 | 6 |
| Length of runs wrt median | 5 | 1 | 11 | 10 | 3 | 3 | 3 | 3 | 21 | 3 | 3 | 5 |
| Average collision | 6 | 0 | 42 | 6 | 0 | 11 | 6 | 0 | 6 | 14 | 0 | 6 |
| Maximum collision | 21 | 5 | 1 | 4 | 2 | 4 | 10 | 3 | 3 | 4 | 2 | 81 |
| Compression | 11 | 0 | 6 | 6 | 0 | 61 | 10 | 0 | 6 | 15 | 0 | 6 |
| Periodicity-1 | 16 | 0 | 6 | 9 | 0 | 6 | 6 | 0 | 8 | 6 | 0 | 8 |
| Periodicity-2 | 34 | 0 | 6 | 6 | 0 | 10 | 6 | 0 | 10 | 6 | 0 | 6 |
| Periodicity-8 | 6 | 0 | 6 | 6 | 0 | 112 | 22 | 0 | 6 | 18 | 0 | 6 |
| Periodicity-16 | 48 | 0 | 6 | 6 | 0 | 15 | 20 | 0 | 6 | 6 | 0 | 15 |
| Periodicity-32 | 6 | 0 | 13 | 6 | 0 | 14 | 6 | 0 | 48 | 6 | 0 | 7 |
| Covariance-1 | 7 | 0 | 6 | 16 | 0 | 6 | 18 | 0 | 6 | 6 | 0 | 23 |
| Covariance-2 | 6 | 0 | 113 | 6 | 0 | 44 | 17 | 0 | 6 | 6 | 0 | 7 |
| Covariance-8 | 18 | 0 | 6 | 6 | 0 | 22 | 30 | 0 | 6 | 6 | 0 | 41 |
| Covariance-16 | 6 | 0 | 289 | 6 | 0 | 55 | 9 | 0 | 6 | 6 | 0 | 9 |
| Covariance-32 | 6 | 0 | 6 | 75 | 0 | 6 | 6 | 0 | 18 | 8 | 0 | 6 |
| Chi-square independence | *p*-value = 0.283383 | | | *p*-value = 0.570821 | | | *p*-value = 0.828947 | | | *p*-value = 0.524847 | | |
| Chi-square goodness of fit | *p*-value = 0.733166 | | | *p*-value = 0.156913 | | | *p*-value = 0.427471 | | | *p*-value = 0.665661 | | |
| Length of the longest repeated substring | Passed | | | Passed | | | Passed | | | Passed | | |
| Restart | Passed | | | Passed | | | Passed | | | Passed | | |
| Min. entropy per byte | 7.987921 | | | 7.988972 | | | 7.985753 | | | 7.989519 | | |

The second group of tests can be found in the NIST Test Suite SP800-90B [179]. Experimental results obtained for the same TRNGs as before are presented in Table 9.4. It lists, for permutation tests, the counter values saved when a given test was terminated. As can be seen, all TRNGs passed those tests. They also passed four additional statistical tests, as shown in the table. The last row of Table 9.4 reports the min-entropy extracted from the test sequences by native procedures implemented within the SP800-90B suite.

The last group of tests come from the AIS-31 Test Suite [91]. The pass rate corresponding to tests T1 – T5 are reported in Table 9.5. Furthermore, the lower part of the table gives the test statistics obtained for the examined TRNGs (the passing criteria for each test can be found in the second column of the table). As can be seen, the random sequences produced by the new TRNGs pass all statistical tests. In particular, the pass rate for tests T0 – T5 is as high as 100%.

A crucial figure of merit when introducing a new scheme is its logic silicon real estate. As shown earlier, the proposed *n*-bit TRNG requires *n* flip-flops (FF) and a certain number

Table 9.5 Results for binary sequences under AIS-31 tests.

| Test | | $G_{48}$ | $G_{64}$ | $G_{128}$ | $G_{256}$ |
|---|---|---|---|---|---|
| | | Pass rate | | | |
| T0 | Disjointness | Passed | Passed | Passed | Passed |
| T1 | Monobit | 257/257 | 257/257 | 257/257 | 257/257 |
| T2 | Poker | 257/257 | 257/257 | 257/257 | 257/257 |
| T3 | Run | 257/257 | 257/257 | 257/257 | 257/257 |
| T4 | Long run | 257/257 | 257/257 | 257/257 | 257/257 |
| T5 | Autocorrelation | 257/257 | 257/257 | 257/257 | 257/257 |
| | | Test values (Statistics – S) | | | |
| T6 - a | Uniform distr. ( $S < 0.025$) | 0.000580 | 0.003759 | 0.001929 | 0.001879 |
| T6 - b | Uniform distr. ($S < 0.020$) | 0.002950 | 0.000950 | 0.002910 | 0.000099 |
| T7 - a | Comparative multinomial, width = 3 ($S < 15.13$) | 0.132880 | 2.394322 | 0.011520 | 0.158421 |
| | | 0.353780 | 0.064981 | 0.144500 | 2.464020 |
| T7 - b | Comparative multinomial, width = 4 ($S < 15.13$) | 2.888009 | 0.003920 | 0.079380 | 0.121680 |
| | | 0.027380 | 0.474320 | 1.764182 | 0.259920 |
| | | 0.840506 | 0.084503 | 0.095220 | 0.165621 |
| | | 0.019220 | 0.420500 | 0.014580 | 0.985681 |
| T8 | Entropy ($S > 7.976$) | 8.001270 | 7.997012 | 8.001384 | 8.001931 |

(depending on *n*) of XOR gates, plus an even number of inverters, a single 2-input NAND gate, and a reset logic. In addition to these numbers, Table 9.6 reports the silicon real estate taken up by generators in terms of equivalent area of 2-input NAND gates (measured also in $\mu m^2$). The presented numbers were obtained with a commercial synthesis tool. All components of the new TRNG logic were synthesized using a 65nm CMOS standard cell library under 2.5ns timing constraint. The last three columns of the table report the resultant silicon area with respect to combinational and sequential devices, and the total area taken by the circuits. Finally, in order to compare the performance of the new scheme with the existing state-of-the-art solutions, Table 9.7 uses six recent techniques as they already compare favorably with other schemes introduced earlier in the technical literature.

Table 9.6 Hardware footprint – equivalent 2-input NAND gates ($\mu m^2$).

| | Gate count | FF count | Combinational | Sequential | Total |
|---|---|---|---|---|---|
| $G_{48}$ | 69 | 48 | 124 (145.23) | 253 (296.31) | 377 (441.54) |
| $G_{64}$ | 91 | 64 | 163 (190.9) | 337 (394.69) | 500 (585.59) |
| $G_{128}$ | 206 | 128 | 331 (387.66) | 672 (787.04) | 1003 (1174.7) |
| $G_{256}$ | 390 | 256 | 629 (736.68) | 1345 (1575.25) | 1974 (2311.92) |

Table 9.7 Comparison with related works.

| | [190] | [104] | [4] | [31] | [49] | This work |
|---|---|---|---|---|---|---|
| NIST SP800-22 | NA | pass | pass | pass | pass | pass |
| NIST SP800-90B | NA | pass | NA | NA | NA | pass |
| AIS-31 | T0 - T5 | pass | T5 and T8 | NA | pass | pass |
| Hardware footprint | 10 LUT, 5 FF | 53 LUT, 22 FF | 528 LUT, 177 FF | 73 slices | 4 LUT, 3 FF | $n/2$ LUT, $n$ FF* |
| Postprocessing | no | yes for FPGA | yes | not needed | not needed | not needed |
| Bit rate [Mb/s] | 1.15 | 1,600 | 6 | 0.011 | 0.76 | 200 |
| FPGA implementation | yes | yes | yes | yes | yes | yes |

NA – not available,

* $n$ – the ring generator size; the number of LUTs is an upper bound assuming that (1) a LUT can accommodate two 2-input XOR gates, (2) a ring generator requires $n/2$ 2-input XOR gates to implement its feedback network, and (3) a ring oscillator needs $n/2$ 2-input XOR gates to inject its values to every "upper level" flip-flop of the ring generator; since the total number of required 2-input XOR gates is typically much smaller than $n$, the number $n/2$ of LUTs can be regarded as accounting for inverters implementing the ring oscillator as well.

## 9.5 Resilience against attacks

TRNGs relying on ring oscillators must be resilient against various types of attacks, including those that destroy their source of entropy. One of the most eminent in this category is a so-called the frequency-injection attack that is capable of successfully locking TRNG to frequencies injected into the power supply, and thus reducing the entropy of a TRNG, as demonstrated in [109]. Following the procedures discussed in [104] and [109], the frequency injection was applied to four TRNGs presented earlier in this chapter in Tables 9.3, 9.4, and 9.5. In particular, after injecting frequency $0.5f_0$, where $f_0$ is the base frequency of the ring oscillator, one can identify three visible frequency peaks in the examined spectrum, i.e., $f_0$ as well as its second and fourth harmonics. Having the external frequency $0.5f_0$ injected into the power supply rails, test sequences are collected, as reported in the previous chapters. The received data are tested by means of NIST SP800–22, NIST SP800–90B, and AIS-31 test suites. The obtained test results clearly indicate that the test sequences passed all tests; the min-entropy statistics after the frequency-injection attack are reported in Table 9.8. As can be seen, the min-entropy drop is negligible; in the worst case it is no greater than 1.7% of the original value.

Table 9.8 The min-entropy after frequency injection.

| TRNG | Original | Under attack |
|---|---|---|
| $G_{48}$ | 0.993773 | 0.979376 |
| $G_{64}$ | 0.996669 | 0.980079 |
| $G_{128}$ | 0.991438 | 0.987242 |
| $G_{256}$ | 0.993663 | 0.985084 |

Also the robustness of the new TRNG against power and thermal attacks was validated. For instance, a power wasting circuitry is often used to overload the power regulator, and hence to degrade the randomness of TRNGs. In order to control the corresponding experiments, the Xilinx system monitor was used to measure the real-time temperature and voltage values altogether with additional 1,600 RO-based power-wasting devices [106] working at different frequencies. With all these circuits enabled, again the required test sequences are collected. As before with the frequency-injection attack, all three test suites are used to evaluate the binary sequences. No negative impact of harsh conditions on test results was observed as the generated data passed all tests.

Finally, with respect to thermal attacks, test sequences produced by TRNG were collected in conjunction with externally injected heat, which does not impact the supply voltage. Moreover, all tests were repeated for different temperatures. The collected experimental results were subsequently validated against the three test suites confirming that the examined TRNGs were virtually immune to the thermal attacks.

## 9.6 Built-in self-test of the generator

Although certain parts of the generator behave, on purpose, in a highly unpredictable manner, testing of its hardware is carried out in a fully deterministic and digital fashion, as briefly discussed in this chapter. As the root-of-trust principal mission is to secure the IC in general, and to protect its design-for-test (DFT) infrastructure in particular, its own test should be an autonomous procedure that does not require any additional DFT components. Similarly to the approach presented in Chapter 7.4, an LBIST can be used to test the generator. In LBIST, the original circuit is typically appended with additional modules for generation of test patterns and compaction of test responses. However, simplicity of design and its inherent iterative functionality can easily facilitate its self-testing.

The entire LBIST session is mainly based on the challenge generator native functionality with a few minor exceptions. In particular, it can be easily modified into PRPG by disabling the ring oscillator feedback loop. The ring oscillator requires a few additional test patterns, as discussed in the following paragraph. Importantly, pseudorandom test data and faulty effects can easily propagate through the ring generator due to its functionality. In fact, the very same ring generator serves as a MISR and contains a final test result. It observes original outputs of the ring oscillator as well as an additional observation point, as discussed below.

During the first step of a test session, all memory elements of the ring generator must be reset. Similarly to the functional mode, the actual testing requires a number of clock cycles necessary to obtain a nonce. The testing results are then available in the ring generator. While running a test, a simple control decoder is employed to stimulate the ring oscillator. To make it testable, the ring oscillator needs to be redesigned by replacing one of the inverters with a

2-input NAND gate $G_3$, and by adding an auxiliary 2-input OR gate $G_1$, as shown in Fig. 9.6. In principle, the test breaks the ring oscillator feedback loop and applies three different patterns (the red font) to inputs of $G_1$, $G_2$, and $G_3$ to detect all single stuck-at faults within the oscillator and to feed the ring generator with deterministic data. This approach is illustrated in Fig. 9.6. It is worth noting that all nets in the circuit assume both values: 0 and 1. It allows one to excite all stuck-at-1 and stuck-at-0 faults, respectively. The first pattern (001) disables the feedback loop at gate $G_2$, whereas the second pattern (010) does the same at gate $G_3$. The last vector (111) blocks the loop at gate $G_1$ that allows us to detect and observe faults on the inputs and the output of $G_2$. The output of $G_1$ is directly connected to the ring generator to observe a response related to s-a-0 fault affecting the feedback line. Clearly, if one of the decoder outputs (driving gates $G_1$, $G_2$, or $G_3$) is stuck at the non-controlling value, and this fault causes one of these inputs to change from a dominating value to a non-controlling one, then the circuit will oscillate, effectively producing a sequence of erroneous values entering the ring generator.

The BIST procedure described above was fault simulated for all single stuck-at faults within the circuitry every generator of Table 9.1 consists of. The obtained results clearly confirm that all single stuck-at faults can be detected within duration of the proposed functional test, i.e., a time needed to produce a nonce.

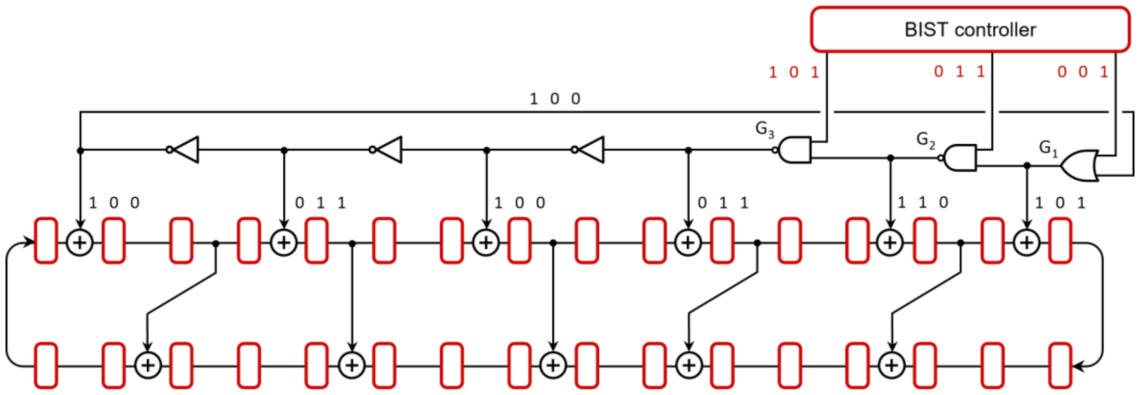


Figure 9.6 Testing a TRNG.

# 10. Hardware root of trust

This chapter introduces a low-cost hardware root of trust to guard ICs security sensitive assets [142]. The infrastructure required is very minimal; the presented scheme employs only a few blocks discussed in previous chapters. In particular, the proposed design deploys customized versions of a challenge generator, a hash function, and input/output SCs, that were presented in previous chapters.

## 10.1 SSN and its root of trust

In principle, the SSN [40] allows one to test, in parallel, any number of cores in a very short time by enabling high-speed data distribution and efficient handling of imbalances between successive modules. Although SSN solves many scan data distribution challenges in large SoC or 3D designs, it may pose security risks, and therefore it needs to be protected and available only to authorized users. Consequently, this chapter proposes to secure the SSN technology by adding a die-centric hardware root of trust protecting SSN-based designs against unauthorized access and expanding threatscape. As SSN is compatible with a flexible parallel port of IEEE Std 1838 for 3D test access, RoT builds on SSN and takes advantage of its central DFT entry to protect a single top level test access point shared by IEEE 1687 (IJTAG) compliant IP blocks. In 3D ICs, the proposed RoT can be either assigned to every silicon wafer or to a master die only.

A simple example of this approach, applied to a 6-core SoC design using SSN, is shown in Fig. 8.1. Each core contains a Streaming Scan Host (SSH) driving local scan resources to load and unload scan chains (or channels) with data delivered on the SSN bus. For example, SSH can interface with EDT logic, as shown in the figure. Typically, each SSH has two external ports: an IEEE 1687 IJTAG interface and a parallel data bus conveying the payload scan data and connecting one SSH to the next one. A single-bit IJTAG network is usually used to configure all SSN nodes prior to application of test patterns. As a result, each node is preloaded with data regarding the active bus width, its location in the series of nodes driven, the number of shift cycles per scan pattern, and other information needed to track the streaming operations. Following this setup, test patterns are applied as packetized scan data that are streamed through the SSH nodes, each of which can determine when it needs (1) to read scan in data from the bus, (2) to place scan out data on the bus, or (3) to pass along data that is destined for other nodes.

The presented SSN modus operandi and a range of data consumed and produced by the SSN nodes are diverse enough to merit the presence of an input SC and an output SC (see Fig. 8.1). Both devices are deployed to decrypt and encrypt the test data from the IJTAG interface and the SSN bus. As a result, they form, in conjunction with the RoT controller and its access authentication mechanisms, effective barriers that may obscure many control and data signals, and thus prevent a wide spectrum of attempts to compromise a design. Indeed, in the case of unauthorized access, randomly obfuscated test data produced by both the input and output SC and sent through the IJTAG and SSN interfaces will cause the entire SoC design to enter an unusual test mode. In this mode, the DFT logic architecture becomes completely unpredictable, leaving the attacker confused or given a fake feedback. Moreover, the same signals may trigger other internal on-chip mechanisms [29], [181] which do not allow normal IP behavior. Architectural details regarding the input (output) SC can be found in Chapter 8.

## 10.2 Challenge-response protocol

The authentication protocol presented in this chapter (and shown in Fig. 10.1) lays foundation for a hardware root of trust. Essentially, it is a small and simple finite-state machine designed to perform a specific set of limited functions like true random number generation, data hashing and encryption, keys validation, and logic locking. The challenge-response procedure works as follows.

Once a request to run certain test-related functions is received, an IC creates a random token (commonly known as a *nonce)* and sends it to a secure server, further referred to as the security processor. The nonce is produced by an on-chip TRNG (see Chapter 9) which yields different strings of 0s/1s within each activation. It contains also some individual data from the IC such as its electronic identification number. The security processor computes a hash of the nonce, as described in Chapter 10.3. This step involves a secret key that is used as an initial value for hashing the nonce data. Selection of the actual key is done based on a received design ID. If the design ID is invalid, the security processor still uses a unique and

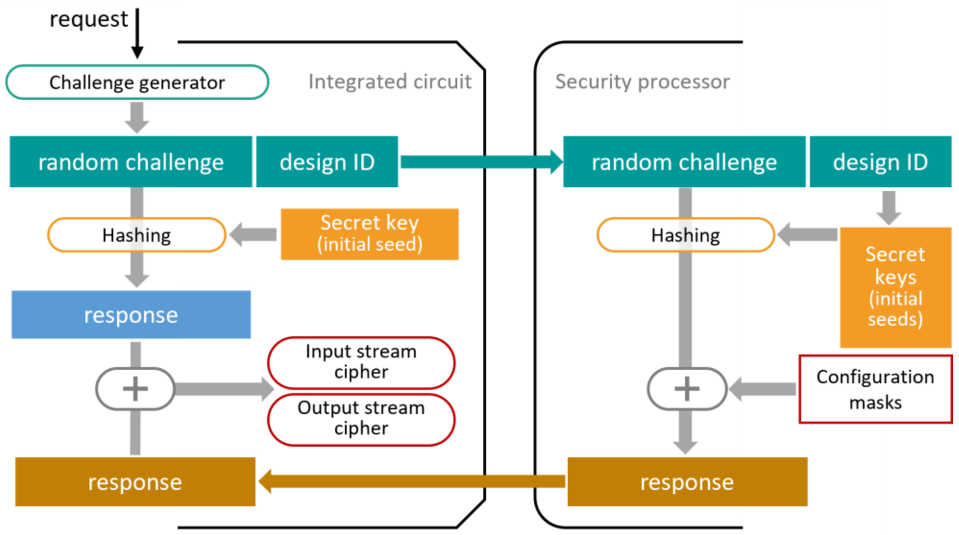


Figure 10.1 The proposed security protocol.

fake (ID-dependent) initial hash value, in this case to obfuscate the resultant response. Furthermore, the security processor may keep track of how many times each individual chip requested a response, monitoring any unusual behavior. The same (valid) secret key is kept in an encrypted form by the IC that hashes the nonce as well, using the secret key the same way the security processor does.

The hash value produced by the security processor is further bitwise XOR-ed with two configuration masks, other secret items stored on the server as a part of a given (legitimate) test set authentication data. Blended with the hash value, they form a response which is sent back to the IC. In order to retrieve the configuration masks, the circuit does a bitwise XOR on the response, i.e., the hash value returned by the security processor and a hash value produced by the IC. They are subsequently used to setup both a test data input and output SC. The former device can handle encrypted test data such as EDT test patterns prior to their further processing by means of on-chip test data decompressors. This way the proposed scheme not only protects the IC through the silicon-based authentication procedure, but also allows one to work with encrypted patterns to mitigate oracle-less attacks targeting directly test data. The same rules apply to the test data output SC that facilitates encrypting of output test data streams. If an attempt of unauthorized access is launched, it triggers changes in the circuit internal functionality. Input and output SCs become blurred due to corrupted configuration masks. This results in signal corruptions caused by activation of certain elements, typically disabled and transparent in the mission mode. The above authentication process is fully automated and invisible to the user (clearly, it does not preclude additional countermeasures such as additional password-based authentication).

A high-level diagram of the proposed root of trust is given in Fig. 10.2. Two devices implementing a random challenge generator (CG) and a hash function sit at the heart of the scheme. A ring-generator-based CG uses a group of ring oscillators. After a given number

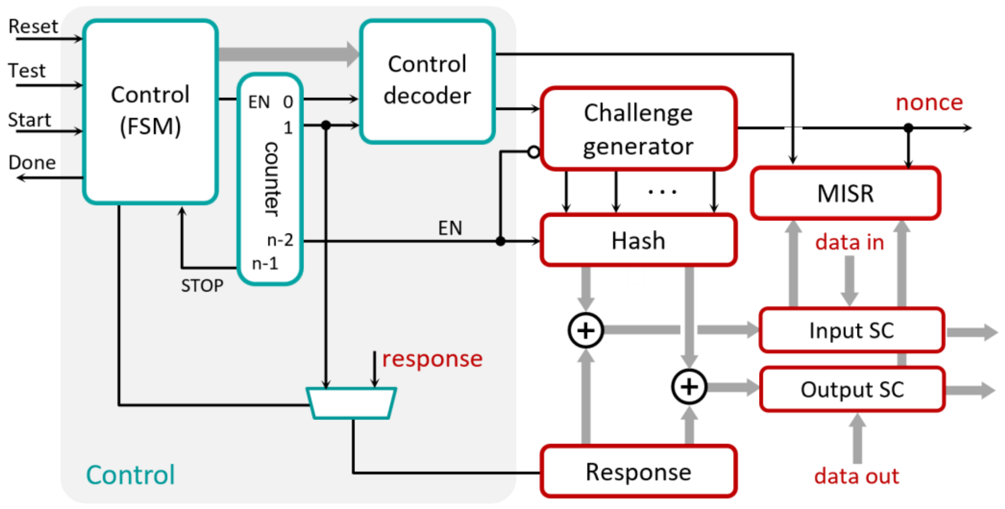


Figure 10.2 Hardware root of trust.

of clock cycles (note that RoT uses its own on-chip clock to avoid noninvasive playing with the clock signals), it serially outputs a sequence of bits that form a nonce going to the security processor. The same nonce becomes now the subject of hashing that mimics the operations carried out by the processor (Chapter 10.3). Activity periods of both devices are controlled by a standard *n*-bit counter, as shown in Fig. 10.2. The same counter signals the main control unit when its most significant output bit changes from 0 to 1. It terminates all operations. RoT deploys also a group of 2-input XOR gates to restore the configuration masks and to feed an input SC, output SC, and obfuscation logic (optional). One group of these gate inputs are driven by a shift register that receives a response produced by the security processor. While a simple FSM implements the control unit and supervises the authentication process, some additional modules such as the control decoder, the MISR, and a 2-way multiplexer as well as some extra connectivity are primarily intended for use in the built-in self-test mode.

A lightweight TRNG presented in Chapter 9 serves as a CG. While the TRNG is driven by an on-chip clock generator, a feedback disable circuitry allows the scheme to convert the CG into a simple shift register when its content is about to be passed to the security processor as a nonce. Typically, it occurs after a predefined number of clock cycles indicated by the counter (Fig. 10.2).

## 10.3 Nonce hashing

As discussed in Chapter 10.2, a random nonce produced by a circuit is subsequently hashed on chip as well as by the security processor that, to deliver a response, runs an appropriate procedure mimicking a built-in hash function of the device-under-test. A hashing circuitry has been presented in Chapter 7. Selection of a particular hash function is decided on the basis of the size message and digest register assume. The final response becomes ready after a predefined number of clock cycles that suffice to rotate the nonce multiple times within the circular register. The main counter of the scheme controls this process.

At the end of the authentication process, the comparator outputs a result that can be further split into three parts, as shown in Fig. 10.2. The first group of output bits form a decryption mask, which was earlier blended into a response (hash), as presented in Chapter 10.2. Upon successful authentication, the mask in its original (proper) form is used to setup an additive input SC. Its modular and programmable feedback network that allows one to implement any characteristic polynomial has been presented in Chapter 8. This, in turn, allows one to pick a suitable secret configuration mask, preferably corresponding to a primitive polynomial, depending on other security needs. The pseudorandom sequences produced by selected polynomials are employed to decrypt, i.e., complement or retrieve, test data that enters the circuit's DFT infrastructure (clearly, this approach requires a prior additive encryption of test data by using the same feedback polynomial). The input SC is initialized by asynchronously setting or resetting its memory elements during the overall RoT reset step. The input SC initial state remains a proprietary information associated with a given design. The second group of output bits form an encryption mask. It is employed to operate the output SC by following the same principles as those described above.

A different scenario applies if the response does not match what is expected, that is, an attempt to unauthorized access is detected. First of all, the decryption mask will trigger a peculiar feedback polynomial that is going to yield a pseudorandom sequence (even not necessarily a maximum-length on its own) that can effectively blur encrypted test data such that it becomes completely useless. The output SC will obscure test results following the same principles.

## 10.4 Security analysis and evaluation

At this point, it is essential to observe that in the case of unauthorized access the proposed authentication scheme does not openly deny access to the attacker by default. Instead, it moves a design (circuit) into a peculiar mode in which its behavior becomes completely unpredictable and misleading. Although it still accepts streams of input data and produces results, in reality this mechanism gives fake feedback to the attacker who may believe to have successfully opened the circuit. In fact, this process rises the complexity of exploration algorithms enormously, also due to characteristic data scrambling features of combined compression hardware as well as SSN and IJTAG protocols. Furthermore, it highly confuses the attacker who is not aware of the actual SoC network configuration and the corresponding operational conditions that they have to deal with.

Having recalled this behavior, consider two basic scenarios. The first one assumes that the attacker has no access to the security processor. It forces the attacker to mimic a response that a circuit should receive after producing a nonce. Assuming that the cryptographic primitives (CG, the hash function) are secure and invulnerable to side-channel attacks, and the secret data kept by the security processor (initial hash values, configuration masks) are secure as well, the resultant security depends primarily on the size of such items as the nonce, the hashed response, and the aforementioned secrets. It is also worth noting that the scheme assures both secure authentication and communication. Indeed, even if the adversary takes over the chip after it has been properly setup by an authorized entity, its usage requires the knowledge of how to encrypt the input data and how to interpret data produced by the circuit. Clearly, that secret information remains in the exclusive possession of the security server.

According to another scenario, a malicious party may try to intercept communication between the IC and the security processor to collect some exemplary data. This case resembles, to some extent, a man in the middle attack. However, the CG used by the proposed scheme to produce a nonce mitigates significantly attempts to discover and resolve challenge-response pairs as the attacker is incapable of guessing the correct value of the nonce in advance. Furthermore, a valid response depends on secret (and secure) data related to the circuit

ID and configuration masks representing the legitimate test patterns (the attacker will likely have to use different patterns that are necessary to compromise internal assets of the IC). In other words, the attacker can eavesdrop on the communication channel but is unable to tamper with it. Clearly, attacks in which the adversary gets a desired device configuration that can be further explored by means of forging patterns are prohibitively expensive if not impossible given (again) large enough secrets. In the light of the above findings, several types of attacks presented in the technical literature [43], including replay attacks, a challenge forgery, response and challenge brute force attacks, are virtually unfeasible.

As shown in [20], the access protocol-based security protection techniques are somehow orthogonal to schemes intended to combat invasive attacks such as microprobing or reverse engineering, and noninvasive attacks (side-channel analysis or fault injection). Consequently, depending on a desired security degree, one may consider additional countermeasures to detect voltage stress, extreme temperatures, clock instabilities, to name just a few anomalies that could be used by malicious parties.

# 11. Conclusions

What has been presented in the thesis clearly supports the observation that deterministic in-system tests can be effectively and safely introduced into the SoC realm. The first part of this work shows that novel in-system test solutions can have unknown states filtered out in a cost-effective manner before they could reach test response compactors. The synergistically combined schemes of the second part of the thesis are capable of creating a customized, lightweight, hardware root of trust for DIST- and SSN-based applications.

In a brief summary, it is worth recalling that the tunable X-tolerant compactor for LBIST applications, presented in Chapter 3, builds on a generic and test set independent scan chain selection technology. It allows one to block, in a highly selective manner, X states within redefinable groups of chains and scan shift cycles. It is also capable of handling observation scan chains that capture test responses in a per-cycle fashion. To the best of the author's knowledge, the X-masking scheme introduced in Chapter 4 is the first solution of that kind developed for DIST applications. The new scan selection logic can be paired with any test response compactor while working with EDT-encoded test data. Both solutions offer very good error observability even in the presence of a large number of unknown states. Hence, the proposed solutions make many in-field and in-system test schemes (such as those developed for automotive electronics) compliant with international test quality standards and very strict quality requirements. Experimental results confirm that X-masking schemes of Chapters 3 and 4 form a robust and superior base for test response compaction techniques that do not impact test quality in terms of test time and test coverage, require a minimal amount of control information, and are easily scalable with the size of tested designs.

Hybrid ring generators, introduced in Chapter 6, can make a substantial contribution toward the performance of linear circuits used in a variety of applications. Similarly to conventional ring generators, the proposed scheme builds on a ring counter and can preserve the maximum-length property. Due to their feedback nets of opposite directions, however, the hybrid rings feature improved structural properties and enhanced the overall performance. In particular, the hybrid rings preserve small internal fan-outs of their conventional counterparts, while simplifying the resultant circuit layout and routing. Consequently, HRGs create a new and enlarged implementation domain for linear finite state machines of a given size and a desired set of feedback taps. HRGs can be also successfully used as building blocks of hardware security primitives, as has been shown in the following chapters.

Chapters 7, 8, and 9 present lightweight security primitives destined for root of trust applications. The cryptographic hash function of Chapter 7 builds on an HRG, which is further combined with a sequential circuit comprising bent-like functions forming its nonlinear feedback network. The test data stream cipher described in the next chapter is again built around an HRG and two additional linearly filtered NLFSRs. A true random number generator from Chapter 9 employs a ring generator (or a hybrid ring generator) driven by a multiple-output ring oscillator. The presented cryptographic hash function and the test data stream cipher are programmable – this feature allows one to pick a linear feedback network to set the HRG up (out of millions of available feedback functions). Moreover, it enables a proprietary initialization of all sequential parts of those modules. Since devices of Chapters 7, 8, and 9 are easily scalable, they can be rearchitected by resizing their building components so that the resultant schemes meet the desired safety requirements of a particular IC. Several desired features of these solutions were examined with various statistical tests, along with the NIST and AIS test suites. The results show that they fulfill all test requirements and can work across a wide range of sizes. Moreover, all three security primitives can resist various types of attacks, including cryptographic ones.

Finally, the last chapter introduces a hardware RoT combining presented earlier security blocks. It is capable of defending designs against unauthorized access to their embedded test instruments. With the proposed approach, trusted users can be granted direct access to the on-chip DFT infrastructure during manufacturing tests. Upon its completion and after blowing test interface fuses, test logic can only be reached through the described RoT. It can be done either by (1) running a test in the input-only streaming mode, which uses encrypted patterns and reference responses for the on-chip compare approach, or by (2) streaming test data in both directions with access to encrypted responses for the sake of fault diagnosis. As encrypting data going to and coming from a DUT makes eavesdropping ineffective, the new scheme counteracts attacks based on the analysis of applied patterns and received results, and it prevents experimentation with varying stimuli and the resulting responses.

The thesis demonstrates that even in such mature areas as test response compaction and in-system test security, there are improvement opportunities. Moreover, the proposed solutions can also be regarded as starting points for future research directions. For example, the test response compactors of Chapters 3 and 4 can be rearchitected to function within the ATE-based test framework, where it is unnecessary to filter out all unknown values. New NLFSRs can be used to further enhance nonlinear sequential logic in hash functions of Chapter 7, or to design fully nonlinear stream ciphers similar to that of Chapter 8. Furthermore, the hardware root of trust discussed in Chapter 10 can be strengthened with a protocol providing access to selected test instruments based on the user's privilege level. In summary, the solutions proposed in the thesis address some of the key challenges of the modern, secure VLSI test and help to converge toward an ultimate on-chip test solution with a negligible impact on the design and manufacturing realm.

# 12. Bibliography

[1] B. Acar and S. Ergun, “A reconfigurable random number generator based on the transient effects of ring oscillators,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 9, pp. 1609-1613, Sept. 2020.

[2] M. Alawida, J.S. Teh, D.P. Oyinloye, W.H. Alshoura, M. Ahmad, and R.S. Alkhawaldeh, “A new hash function based on chaotic maps and deterministic finite state automata,” *IEEE Access*, vol. 8, pp. 113163-113174, 2020.

[3] T. Amaki, M. Hashimoto, and T. Onoye, “An oscillator-based true random number generator with jitter amplifier,” in *Proc. IEEE Int. Symp. Circuits and Systems*, 2011, pp. 725–728.

[4] N.N. Anandakumar, S.K. Sanadhya, and M.S. Hashmi, „FPGA-based true random number generation using programmable delays in oscillator-rings,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 3, pp. 570-574, March 2020.

[5] Y. Atobe, Y. Shi, M. Yanagisawa and N. Togawa, "Dynamically changeable secure scan architecture against scan-based side channel attack," in *Proc. ISOCC*, pp. 155-158, Nov. 2012.

[6] J.-P. Aumasson, L. Henzen, W. Meier, M. Naya-Plasencia, “Quark: a lightweight hash,” *Journal of Cryptology*, vol. 26, pp. 313-339, 2013.

[7] S. Banik, A. Chattopadhyay and A. Chowdhury, "Cryptanalysis of the double-feedback XOR-chain scheme proposed in indocrypt 2013," *Int. Conf. Cryptol. India*, vol. 2014, pp. 179-196.

[8] S. Banik and A. Chowdhury, "Improved scan-chain based attacks and related countermeasures," in *Proc. Int. Conf. Cryptol. India*, pp. 78-97, 2013.

[9] R. Baranowski, M.A. Kochte, and H.-J. Wunderlich, “Fine-grained access management in reconfigurable scan networks,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 6, pp. 937-946, June 2015.

[10] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, A. Ferko, B. Keller, D. Scott, B. Koenemann, and T. Onodera, “Extending OPMISR beyond 10x scan test efficiency,” *IEEE Design and Test of Computers*, vol. 19, no. 5, pp. 65–73, Sep./Oct. 2002.

[11] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, and B. Koenemann, “OPMISR: The foundation for compressed ATPG vectors,” in *Proc. ITC*, 2001, pp. 748–757.

[12] L. Bassham, et al., “A statistical test suite for random and pseudorandom number generators for cryptographic applications,” NIST Special Publication, Tech. Rep. 800-22 Rev 1a, 2010.

[13] A. Beirami and H. Nejati, “A framework for investigating the performance of chaotic-map truly random number generators,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 60, no. 7, pp. 446-450, Jul. 2013.

[14] C. Berbain and H. Gilbert, “On the security of IV dependent stream ciphers,” in *Proc. Int. Workshop on Fast Software Encryption*, LNCS, vol. 4593, Springer-Verlag, 2007, pp. 254–273.

[15] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, “On the indifferentiability of the sponge construction,” in *Proc. Int. Conf. Theory and Applications of Cryptographic Techniques*, Springer, 2008, pp. 181–197.

[16] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, “Keccak,” in *Proc. Int. Conf. Theory and Applications of Cryptographic Techniques*, Springer, 2013, pp. 313-314.

[17] L. Blum, M. Blum, and M. Shub, “A simple unpredictable pseudorandom number generator,” *SIAM J. Comput.*, vol. 15, no. 2, pp. 364–383, 1986.

[18] N. Bochard, F. Bernard, V. Fischer, and B. Valtchanov, “True randomness and pseudo-randomness in ring oscillator-based true random number generators,” *Int. J. Reconfig. Comput.*, vol. 2010, Dec. 2010, Art. no. 879281.

[19] U.J. Botero, R. Wilson, H. Lu, M. T. Rahman, M. A. Mallaiyan, F. Ganji, et al., "Hardware trust and assurance through reverse engineering: A survey and outlook from image analysis and machine learning perspectives", arXiv:2002.04210, 2020

[20] D. Brauchler and J. Dworak, "Multi level access protection for future IEEE P1687.1 IJTAG networks," in *Proc. ITC*, 2020, pp. 1-10.

[21] R. Brederlow, R. Prakash, C. Paulus, and R. Thewes, “A low-power true random number generator using random telegraph noise of single oxide-traps,” in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, 2006, pp. 1666-1675

[22] M. Bucci, L. Germani, R. Luzzi, A. Trifiletti, and M. Varanonuovo, “A high-speed oscillator-based truly random number source for cryptographic applications on a smart card IC,” *IEEE Trans. Comput*., vol. 52, no. 4, pp. 403-409, Apr. 2003.

[23] M. Bucci and R. Luzzi, “Design of testable random bit generators,” in *Proc. Cryptograph. Hardw. Embedded Syst*., 2005, pp. 147-156.

[24] D. Bucerzan, M. Craciun, V. Chis, and C. Ratiu, “Stream ciphers analysis methods,” *Int. J. of Computers, Communications & Control*, vol. V, no. 4, pp. 483-489, 2010.

[25] K.M. Butler and T. J. Powell, “System and method for structurally testing integrated circuit devices,” U.S. Patent 5,694,402, Dec. 2, 1997.

[26] Y. Cao, X. Zhao, W. Zheng, Y. Zheng, and C.-H. Chang, “A new energy-efficient and high throughput two-phase multi-bit per cycle ring oscillator-based true random number generator,” *IEEE Trans. Circuits Syst. I, Regular papers*, vol. 69, no. 1, pp. 272-283, Jan. 2022.

[27] C. Carlet, *Boolean Functions for Cryptography and Coding Theory*, Cambridge University Press, Cambridge, 2021.

[28] K. Chakrabarty, "Zero-aliasing space compaction using linear compactors with bounded overhead," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 17, no. 5, pp. 452-457, May 1998.

[29] A. Chakraborty, N.G. Jayasankaran, Y. Liu, I. Rajendran, O. Sinanoglu, A.Srivastava, Y. Xie, M. Yasin, and M. Zuzak, "Keynote: A disquisition on logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, pp. 1952-1972, Oct. 2020.

[30] U. Chandran and D. Zhao, "SS-KTC: A high-testability low-overhead scan architecture with multi-level security integration," in *Proc. VTS*, pp. 321-326, May 2009.

[31] T. Chen, Y. Ma, J. Lin, Y. Cao, N. Lv, and J. Jing, “A lightweight full entropy TRNG with on-chip entropy assurance,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 12, pp. 2431–2444, Dec. 2021.

[32] X. Chen, Z. Lu, G. Qu and A. Cui, "Partial scan design against scan-based side channel attacks," in *Proc. IEEE Int. Conf. Trust Secur. Privacy Comput. Commun*., pp. 1484-1489, Aug. 2018.

[33] A. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet, “A self-timed ring based true random number generator,” in *Proc. Int. Symp. Asynchronous Circuits and Systems*, 2013, pp. 99-106.

[34] A. Cherkaoui, V. Fischer, L. Fesquet, and A. Aubert, “A very high speed true random number generator with entropy assessment,” in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst*., 2013, pp. 179-196.

[35] V. Chickermane, B. Foutz, and B. Keller, “Channel masking synthesis for efficient on-chip test compression,” in *Proc. ITC*, 2004, pp. 452-461.

[36] G.M. Chiu and J. C. M. Li, "A secure test wrapper design against internal and boundary scan attacks for embedded cores," *IEEE Trans. VLSI Systems*, vol. 20, no. 1, pp. 126-134, Jan. 2012.

[37] C. Clark, "Anti-tamper JTAG TAP design enables DRM to JTAG registers and P1687 on-chip instruments," in *Proc. IEEE Int. Symp. Hardware-Oriented Security Trust*, pp. 19-24, 2010.

[38] C.J. Colburn and A. Rosa, *Triple Systems*, Oxford University Press, New York, 1999.

[39] D. Coppersmith, H. Krawczyk, and Y. Mansour, “The shrinking generator,” *Advances in Cryptology*, LNCS, vol 773, Springer, 1994.

[40] J.-F. Côté, M. Kassab, W. Janiszewski, R. Rodrigues, R. Meier, B. Kaczmarek, P. Orlando, G. Eide, J. Rajski, G. Colon-Bonet, N. Mysore, Y. Yin, and P. Pant, “Streaming scan network (SSN): An efficient packetized data network for testing of complex SoCs,” in *Proc. ITC*, 2020, paper 6B.2.

[41] J. Cui, M. Yi, D. Cao, L. Yai, X. Wang, H. Kiang, Z. Huang, H. Qi, T. Ni, and Y. Lu, “Design of true random number generator based on multi-stage feedback ring oscillator,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 3, pp. 1752-1756, March 2022.

[42] D. Czysz, G. Mrugalski, N. Mukherjee, J. Rajski, and J. Tyszer, “On compaction utilizing inter and intra correlation of unknown states,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, pp. 117-126, Jan. 2010.

[43] J. Da Rolt, A. Das, G. Di Natale, M.-L. Flottes, B. Rouzeyre, and I. Verbauwhede, "Test versus security: past and present," *IEEE Trans. Emerging Topics in Computing*, vol. 2, pp. 50-62, March 2014.

[44] I.B. Damgård, “A design principle for hash functions,” in *Proc. Int. Cryptology Conf.*, Springer, 1989, pp. 416–427.

[45] A. Das et al., "PUF-based secure test wrapper design for cryptographic SoC testing," in *Proc. DATE*, pp. 866-869, 2012.

[46] A. Das, B. Ege, S. Ghosh, L. Batina and I. Verbauwhede, "Security analysis of industrial test compression schemes*", IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, pp. 1966-1977, Dec. 2013.

[47] H. Dattatraya, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, “Silent data corruptions at scale,” https://arxiv.org/abs/2102.11245, 2021.

[48] P.P. Deepthi and P.S. Sathidevi, “Design, implementation and analysis of hardware efficient stream ciphers using LFSR based hash functions,” *Computers & Security*, Elsevier, vol. 28, pp. 229–241, 2009.

[49] R. Della Sala, D. Bellizia, and G. Scotti. “A novel ultra-compact FPGA-compatible TRNG architecture exploiting latched ring oscillators,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 3, pp. 1672-1676, March 2022.

[50] K. Demir and S. Ergun, “Random number generators based on irregular sampling and Fibonacci–Galois ring oscillators,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 10, pp. 1718-1722, Oct. 2019.

[51] S.N. Dhanuskodi, A. Vijayakumar, and S. Kundu, “A chaotic ring oscillator based random number generator,” in *Proc. Int. Symp. Hardware-Oriented Security Trust*, 2014, pp. 160-165.

[52] M. Dichtl and J.D. Golić, “High-speed true random number generation with logic gates only,” in *Proc. Cryptograph. Hardw. Embedded Syst*., 2007, pp. 45-62.

[53] I. Dinur and A. Shamir, “Cube attacks on tweakable black box polynomials,” in *Proc. EUROCRYPT*, Springer, 2009, pages 278–299.

[54] H.D. Dixit, L. Boyle, G. Vunnam, S. Pendharkar, M. Beadon, and S. Sankar, “Detecting silent data corruptions in the wild,” https://arxiv.org/abs/2203.08989, 2022.

[55] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schlaffer, “ASCON v2.1: lightweight encryption and hashing,” *J. of Cryptology*, vol. 34, paper no. 33, 2021.

[56] E. Dubrova, “A scalable method for constructing Galois NLFSRs with period $2^n - 1$ using cross-join pairs,” *IEEE Trans. on Inf. Theory*, vol. 59, no. 1, pp. 703-709, Jan. 2013.

[57] E. Dubrova, “A list of maximum period NLFSRs,” *IACR Cryptol. ePrint Arch.*, vol. 2012, pp. 166-174, 2012.

[58] E. Dubrova, “A transformation from the Fibonacci to the Galois NLFSRs,” *IEEE Trans. on Inf. Theory*, vol. 55, no. 11, pp. 5263-5271, Nov. 2009.

[59] E. Dubrova, M. Teslenko, and H. Tenhunen, “On analysis and synthesis of $(n, k)$-non-linear feedback shift registers,” in *Proc. Design, Automation and Test in Europe*, 2008, pp. 1286-1291.

[60] P. Ekdahl and T. Johansson, “A new version of the stream cipher SNOW,” *Selected Areas in Cryptography*, Springer, LNCS 2595, pp. 47-61, 2002.

[61] H. El-Razouk, A. Reyhani-Masoleh, and G. Gong, "New implementations of the WG stream cipher," *IEEE Trans. VLSI Systems*, vol. 22, no. 9, pp. 1865-1878, Sept. 2014.

[62] M. Epstein, L. Hars, R. Krasinski, M. Rosner, and H. Zheng, “Design and implementation of a true random number generator based on digital circuit artifacts,” in *Proc. Cryptograph. Hardw. Embedded Syst.*, 2003, pp. 152-165.

[63] E. Farcot, S. Best, R. Edwards, I. Belgacem, X. Xu, and P. Gill, “Chaos in a ring circuit,” *Chaos, Interdiscipl. J. Nonlinear Sci*., vol. 29, no. 4, Apr. 2019, Art. no. 043103.

[64] V. Fischer and M. Drutarovsky, “True random number generator embedded in reconfigurable hardware,” in *Proc. Cryptograph. Hardw. Embedded Syst.*, 2002, pp. 415-430.

[65] J. Francq, L. Besson, P. Huynh, P. Guillot, G. Millerioux, and M. Minier, “Non-triangular self-synchronizing stream ciphers,” *IEEE Trans. Comput.*, vol. 71, no. 1, pp. 134-145, Jan. 2022.

[66] H. Fredricksen, “A survey of full length nonlinear shift register cycle algorithms,” *SIAM Review*, vol. 24, no. 2, pp. 195–221, 1982.

[67] B.M. Gammel and R. Göttfert, “Linear filtering of nonlinear shift-register sequences,” in *Proc. Coding and Cryptography*, LNCS, vol 3969, Springer, 2005, pp. 354-370.

[68] B.M. Gammel, R. Göttfert, and O. Kniffler, “An NLFSR-based stream cipher,” in *Proc. Int. Symp. On Circuits and Systems*, 2006, pp. 2917-2920.

[69] J.D. Golić, “New methods for digital generation and postprocessing of random data,” *IEEE Trans. Comput*., vol. 55, no. 10, pp. 1217-1229, Oct. 2006.

[70] G. Gong and A. Youssef, “Cryptographic properties of the Welch-Gong transformation sequence generators,” *IEEE Trans. Infor. Theory*, vol. 48, No. 11, pp. 2837-2846, Nov. 2002.

[71] S.W. Golomb, *Shift* Register *Sequences,* Aegean Park Press, Laguna Hills, California, 1982.

[72] U. Guin, Z. Zhou, and A. Singh, “Robust design-for-security architecture for enabling trust in IC manufacturing and test,” *IEEE Trans. VLSI Systems*, vol. 26, pp. 818–830, May 2018.

[73] X. Guo et al., “Can algorithm diversity in stream cipher implementation thwart (natural and) malicious faults?,” *IEEE Trans. Emerging Topic in Computing*, vol. 4, no. 3, pp. 363-373, July-Sept. 2016.

[74] P. Hagerty and T. Draper, “Entropy bounds and statistical tests,” in *Proc. NIST Random Bit Generation Workshop*, 2012.

[75] C. Hanin, B. Echandouri, F. Omary, and S.E. Bernoussi, “L-CAHASH: A novel lightweight hash function based on cellular automata for RFID,” *Ubiquitous Networking,* Springer, 2017, pp. 287–298

[76] H. Hata and S. Ichikawa, “FPGA implementation of metastability-based true random number generator,” *IEICE Trans. Inf. Syst*., vol. E95.D, no. 2, pp. 426-436, 2012

[77] S. Hellebrand, T. Indlekofer, M. Kampmann, M.A. Kochte, C. Liu, and H.-J. Wunderlich, “FAST-BIST: Faster-than-at-speed BIST targeting hidden delay defects,” in *Proc. ITC*, 2014, paper 29.3.

[78] D. Hely, M. Flottes, F. Bancel, B. Rouzeyre, N. Berard and M. Renovell, "Scan design and secure chip," in *Proc. Int. Line Test. Symp. (IOLTS),* pp. 219-224, 2004.

[79] C. Hobbs and P. Lee, “Understanding ISO 26262 ASILs,” *Electronic Design*, July 9, 2013.

[80] P.H. Hochschild, P. Turner, J.C. Mogul, R. Govindaraju, P. Ranganathan, D.E. Culler, and A. Vahdat, “Cores that don’t count,” in *Proc. Workshop on Hot Topics in Operating Systems*, 2021, pp. 9-16.

[81] D.E. Holcomb, W.P. Burleson, and K. Fu, “Power-up SRAM state as an identifying fingerprint and source of true random numbers,” *IEEE Trans. Comput*., vol. 58, no. 9, pp. 1198-1210, Sept. 2009.

[82] P.D. Hortensius, R. D. McLeod, and H. C. Card, “Parallel random number generation for VLSI systems using cellular automata,” *IEEE Trans. Comput*., vol. 38, no. 10, pp. 1466–1473, 1989.

[83] Y. Huang, S. Milewski, J. Rajski, J. Tyszer, and C. Wang, “Low cost hypercompression of test data,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2964-2975, Oct. 2020.

[84] M. Inoue, T. Yoneda, M. Hasegawa and H. Fujiwara, "Partial scan approach for secret information protection," in *Proc. ETS*, pp. 143-148, May 2009.

[85] S.A. Jassim and A.K. Farhan, “A survey on stream ciphers for constrained environments,” in *Proc. Babylon Int. Conf. on Information Technology and Science*, 2021, pp. 228-233.

[86] A.P. Johnson, R.S. Chakraborty, and D. Mukhopadyay, “An improved DCM-based tunable true random number generator for Xilinx FPGA,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 64, no. 4, pp. 452-456, Apr. 2017.

[87] A. Joux, “Multicollisions in iterated hash functions. Application to cascaded constructions,” in *Proc. Ann. Int. Cryptology Conference*, Springer, 2004, pp. 306-316.

[88] B. Jun and P. Kocher, The Intel random number generator, Cryptography Res. Inc., San Francisco, CA, USA, Apr. 1999.

[89] O. Kara and M. F. Esgin, “On analysis of lightweight stream ciphers with keyed update,” *IEEE Trans. Comput*., vol. 68, no. 1, pp. 99-110, Jan. 2019.

[90] S. Keshavarz, C. Yu, S. Ghandali, X. Xu and D. Holcomb, "Survey on applications of formal methods in reverse engineering and intellectual property protection," *J. Hardw. Syst. Secur.*, vol. 2, no. 3, pp. 214-224, Sep. 2018.

[91] W. Killmann and W. Schindler, "AIS 31: Functionality classes and evaluation methodology for true (physical) random number generators, version 3.1," in *Proc. Bundesamt Sicherheit der Informations-technik* (BSI), Bonn, Germany, 2001, pp. 1-9.

[92] D. Kinniment and E. Chester, "Design of an on-chip random number generator using metastability," in *Proc. Eur. Solid-State Circuits Conf.*, 2002, pp. 595-598.

[93] A. Klein, *Stream Ciphers*, Springer-Verlag, London, 2013.

[94] L. Knudsen and B. Preneel, "Construction of secure and fast hash functions using nonbinary error-correcting codes," *IEEE Trans. Inf. Theory*, vol. 48, no. 9, pp. 2524-2539, Sept. 2002.

[95] P. Kohlbrenner and K. Gaj, "An embedded true random number generator for FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2004, pp. 71-78.

[96] A. Kumar, M. Kassab, E. Moghaddam, N. Mukherjee, J. Rajski, S.M. Reddy, J. Tyszer, and C. Wang, „Isometric test compression," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, pp. 1847-1859, Nov. 2015.

[97] J. Lee, M. Tebranipoor and J. Plusquellic, "A low-cost solution for protecting IPs against scan-based side-channel attacks," in *Proc. VTS*, pp. 6, Oct. 2006.

[98] J. Lee, M. Tehranipoor, C. Patel and J. Plusquellic, "Securing designs against scan-based side-channel attacks," *IEEE Trans. Dependable Secure Comput.*, vol. 4, no. 4, pp. 325-336, Oct. 2007.

[99] R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*, Cambridge University Press, 1994

[100] D. Liu, Z. Liu, L. Li, and X. Zou, "A low-cost low-power ring oscillator-based truly random number generator for encryption on smart cards," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 63, no. 6, pp. 608-612, Jun. 2016.

[101] Y. Liu, S. Milewski, G. Mrugalski, N. Mukherjee, J. Rajski, J. Tyszer, and B. Włodarczak, "X-Tolerant compactor maXpress for in-system test applications with observation scan," *IEEE Trans. VLSI Systems*, vol. 29, no. 8, pp. 1553-1566, Aug. 2021.

[102] Y. Liu, S. Milewski, G. Mrugalski, N. Mukherjee, J. Rajski, J. Tyszer, and B. Włodarczak, "X-Tolerant tunable compactor for in-system test," in *Proc. ITC, 2020,* pp. 1-10.

[103] Y. Liu, G. Mrugalski, N. Mukherjee, J. Rajski, J. Tyszer, and B. Włodarczak, „Universal compactor architecture for testing circuits" U.S. Patent 11,815,555, Nov. 14, 2023.

[104] Y. Luo, W. Wang, S. Best, Y. Wang, and X. Xu, "A high-performance and secure TRNG based on chaotic cellular automata topology," *IEEE Trans. Circuits Syst. I, Regular papers*, vol. 67, no. 12, pp. 4970-4983, Dec. 2020.

[105] M.U. Maaz, A. Sprenger, and S. Hellebrand, "A hybrid space compactor for adaptive X-handling," in *Proc. ITC*, 2019, paper 3.3.

[106] D. Mahmoud and M. Stojilovic, "Timing violation induced faults in multi-tenant FPGAs," in *Proc. DATE*, 2019, pp. 1745–1750.

[107] A. Maiti, R. Nagesh, A. Reddy, and P. Schaumont, "Physical unclonable function and true random number generator: A compact and scalable implementation," *in Proc. ACM Great Lakes Symp. VLSI*, 2009, pp. 425-428.

[108] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA-based true random number generation using circuit metastability with adaptive feedback control," in *Proc. Cryptograph. Hardw. Embedded Syst.*, 2011, pp. 17-32.

[109] A.T. Markettos and S. W. Moore, "The frequency injection attack on ring-oscillator-based true random number generators," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.*, 2009, pp. 317–331.

[110] J.L. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. Inf. Theory*, vol. 15, no. 1, pp. 122-127, Jan. 1969.

[111] M. Matsumoto, S. Yasuda, R. Ohba, K. Ikegami, T. Tanamoto, and S. Fujita, “1200µm$^2$ physical random-number generators based on sin MOSFET for secure smart-card application,” in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2008, pp. 414-624.

[112] W. Meier and O. Staffelbach, “The self-shrinking generator,” *Advances in Cryptology*, LNCS, vol 950, Springer, 1995, pp. 205-214.

[113] R.C. Merkle, “One way hash functions and DES,” in *Proc. Int. Cryptology Conf., Springer*, 1989, pp. 428–446.

[114] S. Mitra and K. S. Kim, “X-compact: An efficient response compaction technique,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 3, pp. 421–432, Mar. 2004.

[115] S. Mitra, S.S. Lumetta, and M. Mitzenmacher, “X-tolerant signature analysis,” in *Proc. ITC*, 2004, pp. 432-441.

[116] A. Mittelbach and M. Fischlin, *The Theory of Hash Functions and Random Oracles: An Approach to Modern Cryptography*, Springer, New York, 2021.

[117] M.H. Moaiyeri, R.F. Mirzaee, K. Navi, T. Nikoubin, and O. Kavehei, “Novel direct designs for 3-input XOR function for low-power and high-speed applications,” *Int. Journal of Electronics*, vol. 97, no. 6, pp. 647-662, 2010.

[118] E. Moghaddam, N. Mukherjee, J. Rajski, J. Solecki, J. Tyszer, and J. Zawada, “Logic BIST with capture-per-clock hybrid test points,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 6, pp. 1028–1041, Jun. 2019.

[119] G.E. Moore, “Cramming more components onto integrated circuits,” *Electronics*, vol. 38, no. 8, pp. 114-117, April 1965.

[120] G. Mrugalski, N. Mukherjee, J. Rajski, and J. Tyszer, “High performance dense ring generators,” *IEEE Trans. Comput.*, vol. 55, no. 1, pp. 83-87, Jan. 2006.

[121] G. Mrugalski, J. Rajski, and J. Tyszer, “Ring generators – new devices for embedded test applications,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst..*, vol. 23, no. 9, pp. 1306-1320, Sep. 2004.

[122] G. Mrugalski, J. Rajski, J. Tyszer, and B. Włodarczak, "X-Masking for deterministic in-system tests," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 11, pp. 4260-4269, Nov. 2023.

[123] N. Mukherjee et al., “Time and area optimized testing of automotive ICs,” *IEEE Trans. VLSI Systems*, vol. 29, no. 1, pp. 76–88, Jan. 2021.

[124] N. Mukherjee, A Pogiel, J. Rajski, and J. Tyszer, “High-speed on-chip event counters for embedded systems”, in *Proc. Int. Conf. on VLSI Design*, 2009, pp. 275-280.

[125] N. Mukherjee, J. Rajski, G. Mrugalski, A. Pogiel, and J. Tyszer, “Ring generator: an ultimate linear feedback shift register,” *IEEE Computer*, vol. 44, no. 6, pp. 64-71, June 2011.

[126] B. Nadeau-Dostie, “Method of masking corrupt bits during signature analysis and circuit for use therewith,” U.S. Patent 6,745,359, Jun. 1, 2004.

[127] P. Nannipieri, S. Di Matteo, L. Baldanzi, L. Crocetti, J. Bell, L. Fanucci, and S. Saponara, “True random number generator based on Fibonacci-Galois ring oscillators for FPGA,” *Appl. Sci.*, vol. 11, 3330, 2021.

[128] M. Naruse, I. Pomeranz, S.M. Reddy, and S. Kundu, “On-chip compression of output responses with unknown values using LFSR reseeding,” in *Proc. ITC*, 2003, pp. 1060-1068.

[129] F. Novak and A. Biasizzo, "Security extension for IEEE Std 1149.1,” *J. Electron. Test. Theory Appl.*, vol. 22, no. 3, pp. 301-303, 2006.

[130] J.H. Patel, S. S. Lumetta, and S. M. Reddy, “Application of Saluja-Karpovsky compactors to test responses with many unknowns,” in *Proc. VTS*, 2003, pp. 107–112.

[131] A. Peetermans, V. Rozić, and I. Verbauwhede, “A highly-portable true random number generator based on coherent sampling ,” in *Proc. IEEE Int. Conf. on Field Programmable Logic and Applications*, 2019, pp. 218-224.

[132] L. Petrica, “FPGA optimized cellular automaton random number generator,” *J. Parallel Distrib. Comput.*, vol. 111, pp. 251-259, Jan. 2018.

[133] C.S. Petrie and J. A. Connelly, “A noise-based IC random number generator for applications in cryptography,” *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 47, no. 5, pp. 615-621, May 2000.

[134] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, “A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices,” in *Proc. Int. Conf. Field Program. Logic Appl.*, 2016, pp. 1-10

[135] M.A. Philip and Vaithiyanathan, “A survey on lightweight ciphers for IoT devices,” *in Proc. Int. Conf. on Technological Advancements in Power and Energy*, 2017, pp. 1-4.

[136] S. Pilarski and T. Kameda, *Probabilistic Analysis of Test-Response Compaction*, IEEE Computer Society Press, Los Alamitos, CA, 1995.

[137] I. Pomeranz, S. Kundu, and S.M. Reddy, “On output response compression in the presence of unknown output values,” in *Proc. DAC*, 2002, pp. 255-258.

[138] B. Preneel, “Davies–Meyer hash function,” in *Encyclopedia of Cryptography and Security*, Boston, Springer, 2005, p. 136

[139] J. Rajski, M. Trawka, J. Tyszer, and B. Włodarczak, "$H_2B$: Crypto hash functions based on hybrid ring generators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 2, pp. 442-455, Feb. 2024.

[140] J. Rajski, M. Trawka, J. Tyszer, and B. Włodarczak, “Hybrid ring generators for in-system test applications,” in *Proc. ETS*, 2023, pp. 1-6.

[141] J. Rajski, M. Trawka, J. Tyszer, and B. Włodarczak, “A lightweight true random number generator for root of trust applications,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*., vol. 42, no. 9, pp. 2815-2825, Sept. 2023.

[142] J. Rajski, M. Trawka, J. Tyszer, and B. Włodarczak, “Hardware root of trust for SSN-based DFT ecosystems”, in *Proc. ITC*, 2022, pp. 450-454.

[143] J. Rajski and J. Tyszer, “Synthesis of X-tolerant convolutional compactors,” in *Proc. VTS*, 2005, pp. 114–119.

[144] J. Rajski and J. Tyszer, “Primitive polynomials over GF(2) of degree up to 660 with uniformly distributed coefficients,” *Journal of Electronic Testing: Theory and Applications*, vol. 19, pp. 645-657, 2003.

[145] J. Rajski and J. Tyszer, “Automated synthesis of phase shifters for built-in self-test applications,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 10, pp. 1175-1188, Oct. 2000.

[146] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, “Embedded deterministic test,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 5, pp. 776–792, May 2004.

[147] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, “Method for synthesizing linear finite state machines”, US patent 6,353,842, 2002.

[148] J. Rajski, J. Tyszer, G. Mrugalski, W.-T. Cheng, N. Mukherjee, and M. Kassab, “Multi-stage test response compactors,” U.S. Patent 7,818,644, Oct. 19, 2010.

[149] J. Rajski, J. Tyszer, G. Mrugalski, N. Mukherjee, W.-T. Cheng, M. Kassab, “X-Press: two-stage X-tolerant compactor with programmable selector”, *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 1, pp. 147-159, Jan. 2008.

[150] J. Rajski, J. Tyszer, G. Mrugalski, N. Mukherjee, W.-T. Cheng, M. Kassab, M. Sharma, and L. Lai, “X-tolerant compactor with on-chip registration and signature-based diagnosis,” *IEEE Design and Test of Computers*, vol. 24, pp. 476 – 485, Sept.-Oct. 2007.

[151] J. Rajski, J. Tyszer, C. Wang, and S. Reddy, “Convolutional compaction of test responses,” in *Proc. ITC*, 2003, pp. 745–754.

[152] M.T. Rahman, K. Xiao, D. Forte, X. Zhang, J. Shi, and M. Tehranipoor, “TI-TRNG: Technology independent true random number generator,” in *Proc. DAC*, 2014, pp. 1-6.

[153] M.A. Razzaq, V. Singh and A. Singh, "SSTKR: Secure and testable scan design through test key randomization,” in *Proc. ATS*, pp. 60-65, Nov. 2011.

[154] M. Robshaw and O. Billet (eds), *New Stream Cipher Designs, The eSTREAM Finalists*, LNCS, vol 4986, Springer, 2008.

[155] S. Robson, B. Leung, and G. Gong, “Truly random number generator based on a ring oscillator utilizing last passage time,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 12, pp. 937-941, Dec. 2014.

[156] P. Rogaway, “Formalizing human ignorance: collision-resistant hashing without the keys,” in *Proc. VIETCRYPT*, Springer, 2006, pp. 211–228.

[157] J. Roth, “Diagnosis of automata failures: a calculus and a method,” *IBM Journal of Research and Development*, vol. 10, no. 4, pp. 278–291, Jul. 1966.

[158] O.S. Rothaus, “On bent functions,” *Journal of Combinatorial Theory*, Ser. A, vol. 20, no. 3, pp. 300-305, May 1976.

[159] V. Rozić, B. Yang, W. Dehaene, and I. Verbauwhede, “Highly efficient entropy extraction for true random number generators on FPGAs,” in *Proc. DAC*, 2015, pp. 1-6.

[160] D. Schellekens, B. Preneel, and I. Verbauwhede, “FPGA vendor agnostic true random number generator,” in *Proc. Int. Conf. Field Programmable Logic and Applications*, 2006, pp. 1-6 .

[161] B. Schneier, *Applied Cryptography, Protocols, Algorithms, and Source Code in C*, Second edition, John Wiley and Sons, New York, 1996.

[162] R.A. Schulz, “Random number generator circuit,” US Patent 4,905,176, Feb. 27, 1990.

[163] J. Seberry and X.-M. Zhang, “Highly nonlinear 0-1 balanced Boolean functions satisfying strict avalanche criterion,” *Advances in Cryptography*, vol. 718, LNCS, Springer, pp. 145-155, 1993.

[164] G. Sengar, D. Mukhopadhyay, and D. R. Chowdhury, ‘‘Secured flipped scan-chain model for crypto-architecture,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*., vol. 26, no. 11, pp. 2080–2084, Nov. 2007.

[165] K. Serebryany, M. Lifantsev, K. Shtoyk, D. Kwan, and P. Hochschild, “SiliFuzz: Fuzzing CPUs by proxy,” https://arxiv.org/abs/2110.11519, 2021.

[166] M. Sharma and W.-T. Cheng, “X-filter: Filtering unknowns from compacted test responses,” in *Proc. ITC*, 2005, pp. 1–10, paper 42.1.

[167] N. Sklavos, R. Chaves, G. Di Natale, and F. Regazzoni (eds.), *Hardware security and trust*, Springer, New York, 2017.

[168] C. Srinivasan, K.V. Lakshmy, and M. Sethumadhavan, “Measuring diffusion in stream ciphers using statistical testing methods,” *Defense Science Journal*, vol. 62, no. 1, pp. 6-10, Jan. 2012.

[169] A. Stefanov, N. Gisin, L. Guinnard, and H. Zbinden, “Optical quantum random number generator,” *J. Modern Opt.*, vol. 47, no. 4, pp. 595-598, 2000.

[170] P. Subramanyan, N. Tsiskaridze, W. Li, A. Gascón, W. Y. Tan, A. Tiwari, N. Shankar, S. A Seshia, and S. Malik, “Reverse engineering digital circuits using structural and functional analyses” *IEEE Transactions on Emerging Topics in Computi*ng , vol. 2, no. 1, pp. 63-80, March 2014.

[171] B. Sunar, W. J. Martin, and D.R. Stinson, “A provably secure true random number generator with built-in tolerance to active attacks,” *IEEE Trans. Comput.*, vol. 56, no. 1, pp. 109-119, Jan. 2007.

[172] H. Tang, C. Wang, J. Rajski, S.M. Reddy, J. Tyszer, and I. Pomeranz, “On efficient X-handling using a selective compaction scheme to achieve high test response compaction ratios,” in *Proc. VLSI Design*, 2005, pp. 59-64.

[173] Y. Tang, H.-J. Wunderlich, P. Engelke, I. Polian, B. Becker, J. Schloffel, F. Hapke, and M. Wittke, “X-masking during logic BIST and its impact on defect coverage,” *IEEE Trans. VLSI Systems*, vol. 14, no 2, pp. 193-202, Feb. 2006.

[174] M. Techranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*, Springer, New York, 2012.

[175] F. Tehranipoor, P. Wortman, N. Karimian, W. Yan, and J.A. Chandy, “DVFT: A lightweight solution for power-supply noise-based TRNG using dynamic voltage feedback tuning system,” *IEEE Trans. VLSI Systems*, vol. 26, no. 6, pp. 1084-1097, June 2018.

[176] T.E. Tkacik, “A hardware random number generator,” in *Proc. Cryptograph. Hardw. Embedded Syst.*, 2002, pp. 450-453.

[177] C. Tokunaga, D. Blaauw, and T. Mudge, “True random number generator with a metastability-based quality control,” *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 78-85, Jan. 2008.

[178] N.A. Touba, “X-canceling MISR – An X-tolerant methodology for compacting output responses with unknowns using a MISR,” in *Proc. ITC*, 2007, paper 6.2.

[179] M.S. Turan et al., “Recommendation for the entropy sources used for random bit generation,” NIST Special Publication, Tech. Rep. 800-90B, 2018.

[180] M.S. Turan, A. Doganaksoy, and C. Calık, “Statistical analysis of synchronous stream ciphers,” in *Proc. Int. Workshop on Stream Ciphers Revisited*, 2006, pp. 84-93.

[181] E. Valea, M. Da Silva, G. Di Natale, M.-L. Flottes and B. Rouzeyre, "A survey on security threats and countermeasures in IEEE test standards", *IEEE Design & Test*, vol. 36, pp. 95-116, May/June 2019.

[182] I. Vasyltsov, E. Hambardzumyan, Y.-S. Kim, and B. Karpinsky, “Fast digital TRNG based on metastable ring oscillator,” in *Proc. Cryptograph. Hardw. Embedded Syst*., 2008, pp. 164-180.

[183] E.I. Vatajelu and G. Di Natale, “High-entropy STT-MTJ-based TRNG,” *IEEE Trans. VLSI Systems*, vol. 27, no. 2, pp. 491-495, Feb. 2019.

[184] E.H. Volkerink and S. Mitra, “Response compaction with any number of unknowns using a new LFSR architecture,” in *Proc. DAC*, 2005, pp. 117–122.

[185] J. von Neumann, “Various techniques used in connection with random digits,” in *Monte Carlo Method*., Washington, DC, USA: Nat. Bureau Stand. Appl. Math., Dec. 1951, pp. 36-38.

[186] L.-T. Wang and E.J. McCluskey, “Hybrid designs generating maximum-length sequences,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 7, no. 1, pp. 91-99, Jan. 1988.

[187] L.-T. Wang and N.A. Touba, “Method and apparatus for hybrid ring generator design,” US Patent 8,949,299, 2015.

[188] L.-T. Wang, N.A. Touba, R.P. Brent, H. Wang, and H. Xu, “High speed hybrid ring generator design providing maximum-length sequences with low hardware cost,” Technical report, Computer Engineering Research Center, The University of Texas at Austin, 2011.

[189] S. Wang, K.J. Balakrishnan, and W. Wei, “X-Block: an efficient LFSR reseeding-based method to block unknowns for temporal compactors,” *IEEE Trans. Comput*., vol. 57, no. 7, pp. 978-989, July 2008.

[190] P.Z. Wieczorek and K. Gołofit, “Dual-metastability time-competitive true random number generator,” *IEEE Trans. Circuits Syst. I, Regular Papers*, vol. 61, no.1, pp. 134–145, Jan. 2014.

[191] S. Windarta, S. Suryadi, K. Ramli, B. Pranggono, and T. Surya Gunawan, “Lightweight cryptographic hash functions: design trends, comparative study, and future directions,” *IEEE Access*, vol. 10, pp. 82272-82294, 2022.

[192] P. Wohl, J.E. Colburn, J.A. Waicukauski, and G.A. Maston, “X-LBIST: X-tolerant logic BIST,” in *Proc. ITC*, 2018, paper 13.1.

[193] P. Wohl, J.A. Waicukauski, R. Kapur, S. Ramnath, E. Gizdarski, T.W. Williams, and P. Jaini, “Minimizing the impact of scan compression,” in *Proc. VTS*, 2007, pp. 67-74.

[194] P. Wohl, J.A. Waicukauski, F. Neuveux, and J.E. Colburn, “Hybrid selector for high-X scan compression,” in *Proc. ITC*, 2012, paper 9.2.

[195] P. Wohl, J.A. Waicukauski, F. Neuveux, and E. Gizdarski, “Fully X-tolerant, very high scan compression,” in *Proc. DAC*, 2010, pp. 362-367.

[196] P. Wohl, J.A. Waicukauski, S. Patel, and A. Amin, “X-tolerant compression and application of scan-ATPG patterns in a BIST architecture,” in *Proc. ITC*, 2003, pp. 727-736.

[197] P. Wohl, J.A. Waicukauski, and S. Ramnath, “Fully X-tolerant combinational scan compression,” in *Proc. ITC*, 2007, paper 6.1.

[198] K. Wold and S. Petrović, “Security properties of oscillator rings in true random number generators,” in *Proc. IEEE Int. Symp. Design Diagn. Electron. Circuits Syst,*, 2012, pp. 145–150.

[199] K. Wold and C.H. Tan, “Analysis and enhancement of random number generator in FPGA based on oscillator rings,” in *Proc. Int. Conf. Reconfig. Comput. FPGAs*, 2008, pp. 385-390.

[200] B. Yang, V. Rožic, M. Grujic, N. Mentens, and I. Verbauwhede, “ES- TRNG: A high-throughput, low-area true random number generator based on edge sampling,” *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, no. 3, pp. 267-292, Aug. 2018.

[201] B. Yang, K. Wu and R. Karri, "Secure scan: A Design-for-Test architecture for crypto chips," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 10, pp. 2287-2293, Oct. 2006.

[202] J. Yang, Y. Ma, T. Chen, J. Lin, and J. Jing, “Extracting more entropy for TRNGs based on coherent sampling,” in *Proc. Int. Conf. on Security and Privacy in Communication Systems*, Springer, 2016, pp. 694–709.

[203] K. Yang, D. Blaauw, and D. Sylvester, “An all-digital edge racing true random number generator robust against PVT variations,” *IEEE J. of Solid-State Circuits*, vol. 51, no. 4, pp. 1022-1031, Apr. 2016.

[204] G. Yao and U. Parampalli, “Cryptanalysis of the class of maximum period Galois NLFSR-based stream ciphers,” *Cryptography and Communications*, Springer, vol. 13, pp. 847-864, 2021.

[205] N. Yerukala, V. Kamakshi Prasad, and A. Apparao, “Performance and statistical analysis of stream ciphers in GSM communications,” *J. of Communications Software and Systems*, vol. 16, no. 1, pp. 11-18, March 2020.

[206] F.G. Zadegan, U. Ingelsson, E. Larsson, and G. Carlsson, “Reusing and retargeting on-chip instrument access procedures in IEEE P1687,” *IEEE Design and Test of Computers*, vol. 29, no. 2, pp. 79–88, Apr. 2012.

[207] G. Zeng, X. Dong, and J. Bornemann, “Reconfigurable feedback shift register based stream cipher for wireless sensor networks,” *IEEE Wireless Comm. Letters*, vol. 2, no. 5, pp. 559-562, Oct. 2013.