Keyed Logic BIST for Trojan Detection in SoC

Elena Dubrova
School of ICT
Royal Institute of Technology
164 40 Stockholm, Sweden
dubrova@kth.se

Mats Näslund
Ericsson Research
Ericsson AB
164 80 Stockholm, Sweden
mats.naslund@ericsson.com

Gunnar Carlsson
Development Unit Radio
Ericsson AB
164 80 Stockholm, Sweden
gunnar.carlsson@ericsson.com

Ben Smeets
Ericsson Research
Ericsson AB
164 80 Stockholm, Sweden
ben.smeets@ericsson.com

***Abstract*—As demonstrated by the recent attack on Intel's Ivy Bridge processor, the traditional Logic Built-In Self-Test (LBIST) methods do not provide adequate protection of SoC against malicious modifications known as hardware Trojans. In this paper, we introduce a simple but efficient countermeasure against hardware Trojans which exploits non-zero aliasing probability of LBIST. We propose to generate LBIST test patterns based on a configurable key which is decided and programed into the circuit after the manufacturing stage. Since the key and hence expected LBIST signature are unknown at the manufacturing stage, an attack based on selecting suitable values for the Trojan which result in the same signature as a fault-free circuit signature becomes infeasible.**

I. Introduction

Cryptographic methods are used to protect sensitive information against unauthorized modification or accidental disclosure. Cryptographic algorithms providing high assurance exist, e.g. Advanced Encryption Standard (AES) [1]. However, many open problems related to assuring security of a hardware implementation of a cryptographic algorithm remain. Security of a hardware implementation can be compromised by a random hardware fault. For example, if the output of a pseudo-random pattern generator contained in a stream cipher gets stuck to 0, then the stream cipher will be sending messages unencrypted.

To make possible a periodic fault detection in functional circuits during their lifetime, cryptographic systems often employ Logic Built-In Self-Test (LBIST). The traditional testing methods are good at detecting random faults. However, as demonstrated by a recent attack on Intel's Ivy Bridge processor [2], they do not provide adequate protection against malicious circuit modifications known as *hardware Trojans*. The points of modifications can be selected so that the LBIST signature computed for the Trojan-injected circuit is the same as the fault-free circuit signature and thus the Trojan does not trigger LBIST to fail.

Hardware Trojans has been known for a while (also referred to as *sleeper cells*), but previously it was very difficult to inject a Trojan into the supply chain. In today's globalized world where the use of third-party IP from small and relatively new vendors is widespread, this is no longer a problem. It is also easier now to activate a Trojan as wireless connectivity becomes a dominant way of communication. Earlier generations of hardware Trojans had to be activated through a wired network or software [3]. In addition, there is an increasing number of components on a chip which are always on and waiting for a command to wake up other parts of a chip [4].

Apart from Trojans injected into the third-party IP, malicious circuitry can be added during tapeout of a SoC. Today's SoCs contain billions of transistors, so it is very difficult to identify which of them are not a part of the original design. Functional verification is further complicated by the fact that manufacturers are typically given a freedom to add some redundant circuitry to a chip in order to increase the yield [5].

The threat posed by hardware Trojans was recognized by multiple government agencies, e.g. the U.S. Department of Defense. Over the last few years, documents has been published to regulate suppliers of critical components [6]. The discovery of counterfeit chips in safety and security critical industrial and military products [7] exemplified the importance of building protection mechanisms against hardware Trojans.

In this paper, we introduce a simple but efficient countermeasure against hardware Trojans which exploits non-zero aliasing probability of LBIST. We propose to generate LBIST test patterns based on a configurable key which is decided and programed into the circuit after the manufacturing stage. Since the key and hence expected LBIST signature are unknown at the manufacturing stage, an attack based on selecting suitable values for the Trojan which result in the same signature as a fault-free circuit signature becomes possible only if the attacker can guess the key. By making the key large, e.g. 128 bits, such a possibility can be ruled out.

II. Traditional LBIST

Built-In-Self-Test (BIST) was introduced in 80s with the purpose to combat the raising complexity of external testing [8]. In BIST test generation and response capture logic are incorporated on-chip. On-chip circuitry usually works at a much higher frequency than an external tester. So, by embedding the test pattern generator on chip, test application time can be reduced. By embedding the output response analyzer on chip, time to compute the circuit response can be reduced as well.

There are different types of BIST. Logic BIST (LBIST), on which we focus in this paper, is used for testing random digital logic [9]. Memory BIST (MBIST) is designed for testing memories [10].

978-1-4799-6890-9/14/$31.00

The traditional LBIST employs a Pseudo-Random Pattern Generator (PRPG) to generate pseudo-random test patterns that are applied to the Circuit Under Test (CUT) and an output response compactor for obtaining the cumulative value of the output responses of the circuit to these test patterns, called *signature* (see Figure 1). Faults are detected by comparing the computed signature to the expected "good" signature.

Theoretically, it is possible to generate a complete set of test patterns off-line using some Automatic Test Pattern Generation (ATPG) method and store this test set in an on-chip Read Only Memory (ROM). However, such a scheme does not reduce the cost of test pattern generation and requires a very large ROM. Several gigabits of test data may be required for a multi-million gate design [11]. Instead, pseudo-random patterns generated by a Linear Feedback Shift Register (LFSR) are usually used as test patterns [12]. LFSRs are simple, fast, and easy to implement in hardware [13].

The output response compactor is usually implemented by a Multiple Input Signature Register (MISR). Since the output response is compacted, a faulty circuit may produce the same signature as the correct circuit. This is known as an *aliasing* error. If an MISR with a primitive generator polynomial is used[1], then the aliasing probability is bounded by $1/2^n$ [14], where n is the length of the MISR.

LBIST controller contains control circuitry that administrates the LBIST testing process: generation of pseudo-random test patterns, their application to the circuit under test and compaction of responses of the circuit to these test patterns. In operation, the controller initializes the PRPG with the initial state defined by the test initialization parameters. After the initialization, controller counts the number of test patterns generated by the PRPG and stops the PRPG when a predefined number of patterns are generated.

Pseudo-random patterns generated by the PRPG are fed into the circuit under test and propagated through its components. The resulting responses are provided to the MISR. The MISR computes the signature and forwards it to the decision logic.

Decision logic compares the signature computed by the MISR to the known "good" signature to make a decision whether the CUT passed or failed a test cycle of the LBIST. If the MISR signature matches the expected signature, the CUT passes the test; otherwise it fails the test.

The test initialization parameters and the expected signature are stored in a memory or hard-wired during the manufacturing stage. Typically, LBIST is performed automatically at power-up and restart, or in response to an external trigger, e.g., if a hardware or software supervising the chip indicates a fault. In addition, LBIST may be initiated by an operator, e.g., for debugging purposes when a faulty chip is sent for repair.

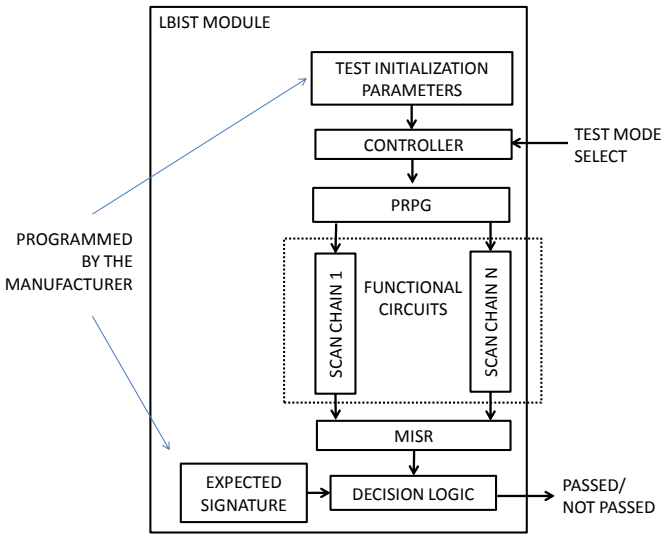

Fig. 1: Traditional LBIST.

III. Hardware Trojans

A hardware *Trojan* is a malicious alteration of a design that makes possible to bypass or disable the security of a system. The purpose of Trojan insertion can be either to leak confidential information to the adversary, or to disable/destroy a chip.

There are two different kinds of Trojans [3]. *Functional* Trojans add or remove transistors, gates or other components to/from the original design. *Parametric* Trojans reduce the reliability of a chip by thinning of wires, weakening of transistors, or subjecting the chip to radiation. A chip with a parametric Trojan produces errors or fails every time the affected component is loaded intensely. While parametric Trojans are always on, functional Trojan are normally in a dormant state. They are triggered either internally by some condition (an internal logic state, a particular input pattern, a counter value) or externally.

Current techniques for Trojan detection include:

1) *Physical inspection* in which the layers of a chip are repeatedly grinded and the exposed circuitry is scanned by various visual inspection methods [15];
2) *Testing* (functional or built-in self), in which test stimuli are applied to a chip and its output is monitored to detect a functional disagreement from the specification [16];
3) *Side-channel analysis* in which signals emitted by a chip, e.g. its leakage current, path delays, or electromagnetic radiation are measured [17], [18].

Some countermeasures have also been developed to protect against activation of certain Trojans, or to maintain secure operation in presence of unknown Trojans [19]. The former typically involves utilizing data guards such as scrambling or obscurification, or hardening the architecture against specific triggers. The latter is usually achieved by replication, fragmentation and voting, as in the traditional fault-tolerant design [20].

[1]An irreducible polynomial of degree n is called primitive if the smallest m for which it divides $x^m + 1$ is equal to $2^n - 1$[13].

IV. Trojans Exploiting Non-Zero Aliasing Probability of LBIST and the Presented Countermeasure

Overall, existing methods for detecting hardware Trojans are still in their infancy. These methods typically focus on a specific class of Trojans, with no single technique being able to provide a complete coverage. For example, the recent attack on the Random Number Generator (RNG) of Intel's Ivy Bridge processor [2] demonstrated that the traditional LBIST may fail even the simple case of functional stuck-at fault type of Trojans. Such type of Trojans can be injected by changing the dopant masks to shorten the outputs of selected gates to GND or to V_{DD}.

Intel's Ivy Bridge processor RNG consists of an entropy source and a digital post-processing unit. The digital post-processing unit contains an Online Health Test (OHT) module and a cryptographically secure Digital Random Bit Generator (DRBG) [21]. The OHT monitors the random numbers for the entropy source to guarantee that they have a required minimum entropy. DRBG includes a conditioner and a rate matcher. The conditioner computes new seeds for the rate matcher. On the based of these seeds, the rate matcher computes 128-bit random numbers.

The RNG is protected by LBIST which checks the functionality of the RNG at each power-up. When LBIST is initiated, the entropy source is disconnected and replaced by a 32-bit LFSR which generates pseudo-random test patterns which are applied to the DRBG. The 32-bit MISR signature representing the compacted output responses of the rate matcher is computed. This signature is compared to a hard-wired expected signature to detect faults in the conditioner and the rate matcher. If two signatures are the same, the RNG passes the LBIST.

Note that, in the traditional LBIST, at each test cycle, the LFSR starts from the same initial state and generates the same number of test patterns which are defined by the test initialization parameters stored on-chip/board. Accordingly, the same set of test patterns is applied to a CUT and the same signature is expected. Therefore, an adversary who knows the set of test patterns generated by the LFSR can made suitable circuit modifications which result in the same signature as a fault-free circuit signature. Fore a 32-bit MISR, the probability that two outputs have the same 32-bit signature is $1/2^{32}$. Therefore, in order to to inject a Trojan in the RNG which does not trigger LBIST, the adversary has to do 2^{31} simulation trials of average.

To demonstrate how such an attack can be done, consider a toy example of a RNG generating numbers in the range $\{1,2,\ldots,15\}$. The RNG can be implemented by a 4-bit Non-Linear Feedback Shift Register (NLFSR) [22] which is updated using the following feedback functions:

$$
\begin{aligned}
&f_0(x_1) = x_1\\
&f_1(x_0,x_1,x_2) = x_2 \oplus x_0x_1\\
&f_2(x_0,x_1,x_3) = x_3 \oplus x_0 \oplus x_1\\
&f_3(x_0) = x_0
\end{aligned}
$$

Test pattern from LFSR $(x_3x_2x_1x_0)$	Fault-free case		With Trojan injected	
	NLFSR response $(x_3x_2x_1x_0)$	MISR signature $(x_3x_2x_1x_0)$	NLFSR response $(x_3x_2x_1x_0)$	MISR signature $(x_3x_2x_1x_0)$
0001	1100	1100	0100	0100
1001	1000	1110	0000	0010
1101	1010	1101	0010	0011
1111	1101	0111	0101	1000
1110	0001	1110	0001	0101
0111	1001	1110	0001	1111
1010	0001	0110	0001	1010
0101	1110	1101	0110	0011
1011	1101	0111	0101	1000
1100	0110	1001	0110	0010
0110	0101	1101	0101	0100
0011	1001	0011	0001	0011

TABLE I: Example.

where x_i represents for the state variable of the ith stage of the NLFSR and f_i is the feedback function of the ith stage, for $i \in \{0,1,2,3\}$.

Suppose that the NLFSR is protected by LBIST in which the PRNG is implemented by a 4-bit LFSR with the generator polynomial $1 \oplus x^3 \oplus x^4$ and the output compactor is implemented by a 4-bit MISR with the generator polynomial $1 \oplus x \oplus x^4$.

An attacker has a chance of $1/2^n$ to correctly guess the number generated by an n-bit NLFSR ($1/2^4$ in our toy example). However, it is possible to inject a Trojan which reduces the complexity of the attack. This can be done by modifying the internal flip-flops of the NLFSR so that they are set to a constant value. If k flip-flops are modified, the complexity of the attack is reduced to $1/2^{n-k}$.

Suppose that the attacker knows that the LFSR generating tests patterns starts from the state (0001) and that 12 tests patterns are generated. The attacker can calculate the expected "good" MISR signature by simulation. From the 3rd column of Table I we can see that in our example the expected signature is (0011). Then, the attacker can compute which signatures are obtained when some of the NLFSR's flip-flops is set to a constant-0 or a constant-1. In our example, the signature (0011) is obtained if the flip-flop corresponding to the stage 3 of the NLFSR is set to 0 (see last column of Table I). So, the attacker can inject the Trojan into the NLFSR and reduce the complexity of the attack by one half. Note that, in general, detecting such a Trojan in a large design using optical reverse engineering is extremely difficult since only the dopant masks of a few transistors have been modified [2]. Since optical reverse engineering is not feasible and the Trojan passes LBIST, a verifier of the design will not be able to distinguish a Trojan-injected circuit from a Trojan-free one. Consequently, the verifier will not be able to provide a "golden" chip (chip which is known to have no malicious modifications). Without a reliably verified golden chip, most post-manufacturing Trojan detection mechanisms [3] cannot be used.

We propose to mitigate this problem by making the initial state of the PRPG dependant on a configurable *key* (see Figure 2). The key determines the set of test patterns generated

by the PRPG and hence the expected MISR signature. The key is decided and programmed into the circuit by the circuit's user after the manufacturing stage. The user also computes the expected signature and programs it into the circuit. Since the expected signature is unknown at the manufacturing stage, an attack based on selecting suitable values for the Trojan which result in the same signature as a fault-free circuit signature becomes possible only if the attacker can guess the key. For a key of size 128-bit, the attacker will need $1/2^{127}$ tries on average to guess the key. Therefore, the attack become infeasible.

The PRPG is adapted to generate test patterns based on an initialization value which is derived from the key. If the key has the same size as the initial state of PRPG, the key itself may be used as an initialization value. Alternatively, if the key is longer than the size of the initial state, the key may be reduced to an initialization value of the same size as the initial state of PRPG by some suitable function (as in the case of keyed PRPG used in cryptographic methods [23]).

The key and the expected signature can be stored in a non-volatile memory, such as a Flash or an Electrically Erasable Programmable Read-Only Memory (EEPROM), or by means of programmable efuses.

Note that the key does not have to be secret. If the currently stored key and the expected signature become compromised, e.g., an adversary gains knowledge of them when the circuit is sent for repair or maintenance, a new key and a new expected signature can be programmed by the user upon receiving the circuit back.

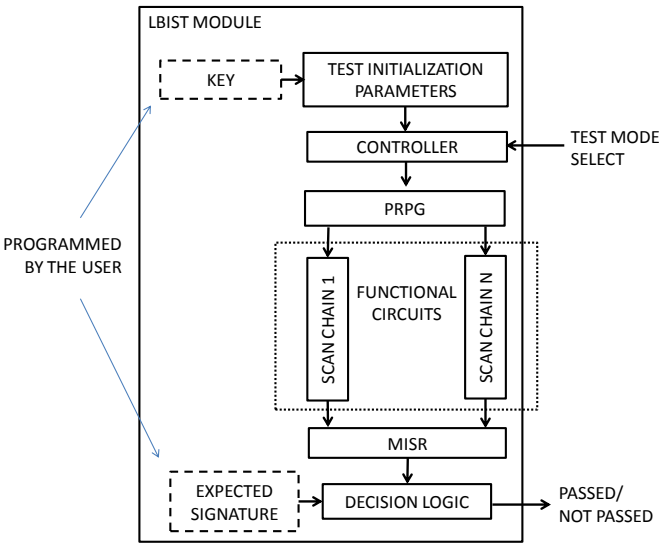

Fig. 2: The presented keyed LBIST.

V. Conclusion

We introduced a countermeasure against hardware Trojans which exploits non-zero aliasing probability of LBIST.

Note that no single method can protect against all possible types of adversarial attacks. Similarly, the proposed method may fail to detect some types of Trojans, e.g. parametric Trojans. Finding adequate protective mechanisms against parametric Trojans remains a topic of future work.

Acknowledgement

The first author was supported in part by the research grant No SM12-0005 from the Swedish Foundation for Strategic Research.

References

[1] J. Daemen and V. Rijmen, "AES proposal: Rijndael," April 2003, national Institute of Standards and Technology.

[2] G. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware Trojans," *Proceedings of Cryptographic Hardware and Embedded Systems (CHES'2013), LNCS 8086*, pp. 197–214, 2013.

[3] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[4] E. Sperling, "The next big threat: Manufacturing," 2014, http://semiengineering.com/manufacturing-and-integration-risks/.

[5] P. Gupta and E. Papadopoulou, "Yield analysis and optimization," in *The Handbook of Algorithms for VLSI Physical Design Automation*. RC Press, 2011.

[6] Department of Defense, "Federal register," Nov. 2013, http://www.steptoe.com/assets/htmldocuments/DFARS

[7] C. Gorman, "Counterfeit chips on the rise," *IEEE Spectrum*, vol. 49, no. 6, pp. 16–17, 2012.

[8] E. McCluskey, "Built-in self-test techniques," *IEEE Design and Test of Computers*, vol. 2, pp. 21–28, 1985.

[9] J. Rajski and J. Tyszer, *Arithmetic Built-In Self-Test for Embedded Systems*. Prentice Hall PTR, 1998.

[10] A. K. Sharma, *Semiconductor Memories: Technology, Testing, and Reliability*. Wiley-IEEE Press, 2002.

[11] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan, and J. Rajski, "Logic BIST for large industrial designs: real issues and case studies," in *Proceedings of International Test Conference (ITC'1999)*, 1999, pp. 358 – 367.

[12] E. McCluskey, S. Makar, S. Mourad, and K. Wagner, "Probability models for pseudorandom test sequences," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 1, pp. 68–74, 1988.

[13] S. Golomb, *Shift Register Sequences*. Aegean Park Press, 1982.

[14] M. Damiani, P. Olivo, M. Favalli, S. Ercolani, and B. Ricco, "Aliasing in signature analysis testing with multiple input shift registers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 12, pp. 1344–1353, 1990.

[15] S. Skorobogatov, "Physical attacks and tamper resistance," in *Introduction to Hardware Security and Trust*, ser. Information Security and Cryptography, M. Tehranipoor and C. Wang, Eds. Springer Berlin / Heidelberg, 2011.

[16] E. Dubrova, M. Näslund, and G. Selander, "Secure and efficient LBIST for feedback shift register-based cryptographic systems," in *Proceedings of International Test Conference (ITC'2014)*, May 2014, pp. 1–6.

[17] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '96. London, UK, UK: Springer-Verlag, 1996, pp. 104–113.

[18] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis." Springer-Verlag, 1999, pp. 388–397.

[19] M. Beaumont, B. Hopkins, and T. Newby, "Hardware Trojans - prevention, detection, countermeasures," Australilan Department of Commerce, Tech. Rep. DSTO-TN-1012, July 2011.

[20] E. Dubrova, *Fault-Tolerant Design*. Springer, 2013.

[21] Intel, "Intel digital random number generator (drng) software implementation guide," Aug. 2012, https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide.

[22] E. Dubrova, "A scalable method for constructing galois NLFSRs with period $2^n - 1$ using cross-join pairs," *IEEE Transactions on Information Theory*, vol. 59, no. 1, pp. 703–709, 2013.

[23] D. Stinson, *Cryptography Theory and Practice*. Chapman & Hall/CRC, 3rd edition, 2006.