

Access provided by:
University of Chinese Academy of SciencesCAS
Sign Out

BrowseMy SettingsGet HelpFull Text

Browse Journals & Magazines > IEEE Transactions on Emerging... > Volume: 5 Issue: 2

A Flexible Online Checking Technique to Enhance Hardware Trojan Horse Detectability by Reliability Analysis

View Document

4Paper Citations

665Full Text Views

Related ArticlesReferences

Fine-line conductor manufacturing using drop-on demand PZT printing technology

New transport services for next-generation SONET/SDH systems

KeywordsView All

5Author(s)

Rajat Subhra Chakraborty ; Samuel Pagliarini ; Jimson Mathew ; Sree Ranjani Rajendran ; M. Nirmala Devi

Footnotes

View All Authors

Back to Top

AbstractAuthorsFiguresReferencesCitationsKeywordsMetricsMedia

Usage

20182017

| Jan | Feb | Mar | Apr | May | Jun |
|-----|-----|-----|-----|-----|-----|
| 29 | 34 | 36 | - | - | - |
| Jul | Aug | Sep | Oct | Nov | Dec |
| - | - | - | - | - | - |

665Total usage since Jan 2017

Best Month: MarYear Total: 99

* Data is updated on a monthly basis. Usage includes PDF downloads and HTML views.

Citations

Scopus®5Web of Science®1

Search for Citations in Google Scholar®

Online Sharing Activity

Powered by Altmetric

Twitter (1)Mendeley (10)

Advertisement

- Download PDF
- Download Citation
- View References
- Email
- Print
- Request Permissions
- Export to Collabratec
- Alerts

SECTION I. Introduction

In recent years, hardware manufacturers are putting faith on business models which are widely based on outsourcing of fabrication facilities and procurement of third-party intellectual property cores (3PIPs). Although such business trends are in sync with current global economic policies, and also improve time-to-market by drastically reducing design and manufacturing cost and time, such practices eventually result in a highly complex supply chain involving thousands of people, as well as complex automated design and manufacturing facilities. As a result, it becomes much easier for attackers (often referred as “adversaries”) to infect the supply chain at various points. Such a breach of security in the supply chain may result in malicious modifications of the integrated circuits (IC), which are referred as *Hardware Trojan Horses* (HTH) [1]– [3]. HTHs are stealthy in nature, made with the intent to evade standard design verification and post-manufacturing tests. In-field activation of HTHs may result in catastrophic system failures or stealing of on-chip secret information, e.g., cryptographic keys. Detection and prevention of HTHs as well as the investigation of newer threats due to them have thus emerged as a prominent area of research.

HTHs consist of two major parts, namely *trigger* and *payload*, both of which can be either analog or digital in nature [2], [3]. The trigger circuit starts the activity of the HTH whereas the payload causes the malfunction. Malfunctions caused by HTHs may be diverse in nature, ranging from the injection of faults like stuck-at-faults [2] to the leaking of secret information through some physical “side-channel”, e.g., power dissipation profile [4]. They may also be parametric causing deterioration of certain physical properties of the chip [5]. The trigger circuits, on the other hand, are specially designed, so that they activate the HTHs only with the occurrence of some extremely

rare logical or physical events in the chip [6]. Such rare triggering helps the HTH to remain hidden during standard design verification and post manufacturing tests. Also, the adversary keeps the size of the HTH circuit small enough to hide its existence amidst the actual circuit logic. A large HTH circuit would most likely incur parametric changes in the circuit, such as a higher than expected current flow and/or power. These could either modify the circuit functionality completely, or be easily detected by monitoring of the circuit. In this paper we expect that the adversary will insert a small, yet smart, piece of logic that cannot be detected through sole analysis of circuit logic functionality or physical characteristics.

Hence, strategies to either simplify HTH detection or to prevent HTH insertion have also become prominent in the recent literature [7], [8]. The authors of [7] performed a vulnerability analysis of circuits at the behavioral level. In this paper, when we present our results, a similar analysis is done at the gate level. This is a significant difference, as it means that our proposed online checking scheme achieves detection rates that are not affected by synthesis and/or optimizations. Prevention schemes, often with significant overhead, try to obfuscate the circuit at different levels of design abstraction by inserting extra circuitry, making the reverse-engineering at the foundry difficult [8], [9]. These techniques can be broadly categorized as *Design-for-Trust* (DFTr) techniques. Segregation of fabrication steps among different foundries, also referred as *Split Manufacturing*, is another proposed solution [1], [10], [11]. Although split manufacturing is realizable at current technology, it requires having a foundry that is willing to pull a wafer halfway through the process, which is not an everyday practice for foundries. Also, runtime detection and prevention methods, which can incur significant overheads, have been proposed. These techniques can incorporate reconfigurable logic along with standard logic [12]. Online checking techniques appear to be a promising option in detecting hard-to-detect HTHs, provided they offer a flexible hardware overhead that can be controlled, making the impact on the circuit performance affordable and acceptable.

In this paper, we propose and implement an automated low-overhead online methodology to aid in the detection of HTHs. Our scheme simultaneously enhances the inherent HTH detectability and reliability of an IC. We concentrate on the detection of small HTH instances (with less than five logic gates) that cause logic malfunction on activation through rare internal logic conditions. Such HTHs are difficult-to-detect through ATPG schemes by design, and for which side-channel analysis schemes are also often ineffective [6], because of their negligible impact on the side-channel signatures. In the proposed scheme, low-overhead checking circuits are inserted at intelligently selected sites inside the netlist, and these checkers can detect and report logic malfunctions in their neighborhood. One of the main advantages of the proposed scheme is that the impact of the activated HTH need not propagate all the way to the primary output for the checker to detect it. This feature ensures that a HTH instance is detected as soon as it is triggered, independent of whether the logic malfunction caused by the HTH actually propagates to the primary output. We describe the online checking algorithm and the associated automated design flow in detail. Experimental results, on ISCAS-85 circuits, establish the effectiveness of the proposed scheme.

The rest of the paper is organized as follows: in Section II, we provide background on HTHs, an overview for Trojan detection techniques (both ATPG-based and other techniques), and establish the motivation for our work. In Section III, we describe our proposed low-overhead online checking methodology, associated design automation flow, and propose a system architecture to follow a “design-for-trust” methodology. Experimental results and advantages of the schemes are described in Section IV. We conclude in Section V with directions for future research.

SECTION II.

Background and Motivation

In this section, we give an overview of HTHs, the state-of-the-art of research on HTH detection/prevention techniques based on test vector generation and Design-for-Trust.

A. Hardware Trojan Horses

Over the years, HTHs have been proposed to be of numerous types, and a complete coverage of all previously proposed HTHs is not within the scope of the work. One of the most common ways of classifying Trojans is based on their activation mechanism and effects. Usually it is assumed that HTHs consist of two parts: a *trigger* for activation and a *payload* to cause the malfunction. However, instances of HTHs without any discernable trigger can also be found [14]. The trigger can be internal (e.g., a large counter or the output of some on-chip sensor [15]), as well as external through some specific input or sequence of inputs. To reduce the chance of accidental activation during testing or in-field operation, the trigger conditions of the HTHs are engineered to be of extremely rare occurrence [6], [16]. A systematic categorization of HTHs, with an emphasis on the HTH structure (i.e., its trigger and payload mechanisms), was presented in [16]. According to this

Full Text

Authors

References

Citations

Keywords

Related Articles

Footnotes

Back to Top

HTH structure (i.e., its trigger and payload mechanisms), was presented in [13]. According to this classification, both digital and analog HTHs may exist. Also, hybrid structures like analog trigger with a digital payload are indeed possible [17]. In this paper, we concentrate on purely digital HTHs, since they are in general the most extensively studied entities in the community.

Figure 1 shows several possible digital HTH trigger and payload mechanisms. The assumption here is that a signal C is going to be modified by a HTH. The digital triggering mechanism may involve only combinational or both combinational and sequential elements. Combinational HTHs involve minimum hardware overheads (as small as a couple of gates), and thus are extremely challenging to detect. To make such small HTHs sufficiently rare to trigger, usually low controllability nets in the circuits are combined so that simultaneous occurrence of the low controllability logic values in them cause the triggering. Figure 1(a) shows the model of such a HTH, which triggers on the simultaneous occurrence of the event $A = B = 0$. Sequential HTHs, on the other hand may have two different forms. The first one is completely synchronous involving a large counter (Figure 1(b)), which triggers when the counter reaches its terminal count (consisting of all counter bits to be logic-1). This is an example of an internally triggered Trojan. Another variation of sequential HTH, also known as asynchronous HTH, is shown in Figure 1(c). In this case, the Trojan waits for some rare sequence(s) to occur multiple times to advance the counter, before triggering when the counter reaches its terminal count. Such models of HTHs cover virtually all possibilities in the digital Trojan space and can be easily extended for hybrid ones (Figure 1(d)). *In this paper, we target the detection of HTHs of the general form shown in Figure 1(a), without loss of generality.* Other detection techniques such as side-channel analysis are much more effective in detecting the types of HTHs containing sequential elements, as exemplified in Figures 1(b), 1(c), and 1(d).



Figure 1

Figure 1.
Structural models of digital HTHs [13], [14].

B. ATPG Techniques for HTH Detection

Testing techniques (either logic testing or side-channel analysis) for HTH detection suffer from several formidable challenges. For logic testing, the main challenge is to ensure the generated test vectors cover the trigger condition(s) of the inserted Trojan(s), which are intentionally designed to be hard to trigger. When performing testing, a specialized algorithm called *Automatic Test Pattern Generation* (ATPG) is used. One must note that the primary goal of ATPG is finding vectors that expose faults, not HTHs. ATPGs are ultimately probabilistic in nature, thereby not being able to guarantee the detection of an inserted HTH [6], [18]. For the side-channel testing based techniques, the main challenge is to isolate the HTH side-channel signature from a noise background that consists of experimental noise and process-variation induced noise in modern nanometer-scale CMOS processes [19]. The problem is exacerbated in circuits that have relatively high operating frequency (e.g., in the GHz range), because of the inherent difficulty of accurate reconstruction of the sampled current or delay signatures.

Logic testing based HTH detection approaches try to generate potential test vectors to trigger the inserted HTHs, as well as to propagate the effect of triggered Trojans effect to some observation point. They are quite attractive for reliable detection of ultra-small HTHs. Moreover, many of the side-channel analysis based approaches for HTH detection assume the availability of specially crafted test vectors to magnify their detection sensitivity; in this process, they often utilize the HTH targeted test vectors generated by ATPG tools, e.g., [20]–[22].

Although test generation for HTH is a very useful and important direction of research, there are several formidable challenges, which have restricted the number of reported ATPG algorithms directed towards HTH detection. As mentioned earlier, due to the inordinately large space of possible Trojans it is not feasible to enumerate all possible HTH instances to generate deterministic test vectors or to compute test coverage. Instead of an exact approach, a sampling based statistical approach for test vector generation can be computationally more tractable. One such method, referred to as *Multiple Excitation of Rare Occurrence* (MERO) was proposed in [6]. The main objective of this technique was to derive a set of test patterns that is compact (minimizing test time and cost), while maximizing the HTH detection coverage. Coverage results obtained after HTH activation (“Trigger Coverage”), and after propagation of logic values (“Trojan Coverage”) were reported for a large set of feasible combinational and sequential HTHs. The main criticism against logic testing based HTH detection approaches is that they do not scale well for large sequential HTHs. Also, it was found that the MERO heuristic has difficulty in detecting techniques with extremely low trigger probability ($< 10^{-3}$). To overcome the shortcomings of MERO, a variation was proposed in [18], that combines some aspects of the MERO heuristic with *Genetic Algorithms* for improved testset generation. Moreover, *Boolean Satisfiability* was used to decrease the size of the feasible Trojan search space. Although this technique obtained some improvements over MERO, it still had several shortcomings, including extremely long computation time even for

[Full Text](#)
[Authors](#)
[References](#)
[Citations](#)
[Keywords](#)
[Related Articles](#)
[Footnotes](#)
[Back to Top](#)

...ate, from hardware shortcomings, making already long comparison time even for moderately sized circuits.

Considering the abovementioned shortcomings, Design-for-Trust techniques have been explored for HTH detection, which we consider next.

C. Design-for-Trust Techniques for HTH Detection/Prevention

Design-for-Trust techniques for HTH detection/prevention can be broadly classified into three classes: (a) to prevent the insertion of HTHs; (b) to assist other Trojan detection schemes, including side-channel analysis, and, (c) to perform online monitoring for HTH detection. We now consider these three cases, with focus on the online HTH detection techniques.

1) Preventive Trust Measures

The best way to prevent the insertion of HTHs is to have a tight control over the IC supply chain from end to end. However, such a control is infeasible to achieve given the modern business and economic trends. Thus, design strategies to achieve certain amount of protection against HTH insertion came into picture.

Preventive trust measures can be classified to be mainly of three types: (a) netlist obfuscation; (b) layout filler, and, (c) supply chain modification. The netlist obfuscation based approaches try to obfuscate the functional and structural properties of the circuit, thereby making it hard to understand and reverse-engineer for an adversary in the fab. The central idea is to incorporate two modes for a circuit through a key. In the obfuscated mode, the operation of the circuit is masked. As a result, it becomes difficult to identify the rare nets which are the potential locations for the insertion of HTHs. The separation between the obfuscated mode and the normal mode is achieved through the modification of the state-machine of the circuit. Till date several circuit obfuscation techniques have been proposed, either for Trojan prevention, or as a standard reverse-engineering countermeasure [8], [9], [23]. Although obfuscation techniques are quite effective, they can add extra hardware overheads. Also, the transition from the obfuscated mode to the normal mode requires a key which is expected to be stored on-chip, which makes these schemes vulnerable to side channel analysis based key recovery attacks [24], [25]. Further, in some recent works it has been shown that even such obfuscated circuits can be reverse-engineered effectively [26].

In the layout filling approaches, the aim is to remove any spaces that could potentially be used for insertion of additional circuitry. This is simply achieved by filling up vacant spaces in the circuit layout. However, an attacker could still identify and replace the filler cells with HTH circuits. *Built-in Self-Authentication* (BISA) is a technique, proposed in [27], to fill all the unused spaces in an IC with functional standard cells instead of filler cells. The self-authenticating structure of BISA prevents any attempt of tampering or changing the BISA architecture without modifying the functionality. BISA provides protection against removal attacks, redesign attacks and resizing attacks. However, it cannot prevent malicious tampering of a set of transistors or addition of a circuit that do not require extra layout space. While challenging, a resourceful adversary may have the power to redesign specific parts of a circuit by means of transistor resizing or constrained logic synthesis to create room for HTH insertion.

As mentioned in Section I, an emerging methodology to prevent Trojan attack relies on splitting the manufacturing process among different foundries [11]. In this approach, the front-end-of-the-line (FEOL) and back-end-of-the-line (BEOL) process steps are divided among separate fabrication facilities to keep the design intent hidden, and to prevent malicious alteration. It considers fabrication of device layers (the FEOL steps) using an advanced process in some untrusted fabrication facility. The higher metal layer interconnect (the BEOL steps) are processed using a trusted fabrication facility. This trusted facility does not necessarily need/have the same lithography process as the FEOL foundry. The lack of the interconnection information to FEOL facilities, prevents an attacker from interpreting a design as well as from incorporating any malicious alteration. However, often it becomes costly to modify the state-of-the-art manufacturing chain, and to establish a “trusted fab”. Since *Multi Project Wafer* (MPW) fabrication runs are not possible with split manufacturing, this solution could incur even more IC fabrication costs, making it almost not commercially viable for low-volume products. Further, security of such schemes is also questionable, as pointed out by some recent works [10].

2) Assistive Trust Measures

Low controllability nets, also called “rare nets” are assumed to be utilized by a HTH attacker for triggering. Thus, removal of low transition nets inside the circuit may reduce the options of the Trojan attacker to insert a HTH. In [7], authors proposed a dummy flip-flop insertion mechanism which, in the test mode increases the transition probabilities of the low transition nets inside the circuit over a predefined threshold (P_{th}). As a result, the inserted HTHs get fully or partially activated more often, facilitating their detection by observing primary outputs or transient current signals. This scheme was further enhanced in [28] by careful insertion of flip-flops, hence reducing the hardware overhead. However, like other Trust mechanisms, these schemes are also vulnerable

Full Text

Authors

References

Citations

Keywords

Related Articles

Footnotes

Back to Top

against intelligent attackers [29], especially, if the P_{th} is not chosen to be significantly high (which may result in significantly high hardware overhead).

3) Online Monitoring Approaches

It is highly desirable that HTHs in the ICs get detected before they are deployed. But due to the fact that state-of-the-art detection and prevention mechanisms cannot guarantee comprehensive HTH detection coverage over all classes of Trojans, online monitoring approaches emerged as the last line of defence. Some works have proposed to add reconfigurable logic to the conventional ASIC flow, so that the reconfigurable blocks can be customized as real-time functionality monitoring units called *security monitors* (SMs) [15], [30]. SMs are finite-state machines (FSM) configured to continuously monitor the behaviour of signals of interest. Such signals are fed to the SM by means of a *signal probe network* (SPN). With the occurrence of some unwanted events, the SMs are assumed to raise some alarm or to initiate suitable countermeasures. SMs are customized to identify unexpected or unwanted behaviour created by a HTH, such as an access to some protected memory space or entering the test mode during normal operation. However, such mechanisms work with an assumption that the SMs or SPNs or their control circuitry cannot be exploited by the Trojan attacker. Software-based monitoring approaches have also been proposed. In microprocessor-based systems, a software solution that can detect HTH activation and/or can tolerate Trojan effects can provide an effective countermeasure. For example, in [31], a verifiable “hardware guard” module, external to the processor, was proposed. It was used for runtime execution monitoring to identify activation of an HTH. It was primarily targeted towards DoS and privilege escalation attacks exploiting periodic checks by the OS, which was enhanced with live check functionality. A similar “memory guard” working concurrently with the memory bus was proposed in [32]. Such countermeasures are again specific to the target architectures and assume unrestricted access to the memory bus. A linear complementary pair (LCP) code based “circuit encoding” technique to detect and prevent small hardware Trojans was presented in [33]. On the occurrence of any logic malfunction at internal node(s) due to the impact of an activated hardware Trojan, the circuit produces invalid codewords, leading to the detection of the Trojan. The authors also provided a CAD methodology, and applied the technique to encode a cryptographic coprocessor. The advantages of the scheme are its tunability, an automated design methodology that can be adopted easily by a hardware designer having little knowledge of security, and its good resistance to side-channel and fault injection attacks. However, its major disadvantage is its exceedingly large hardware overhead, which is over 200 percent for most encoding configurations. Also, it cannot provide 100 percent guarantee of Trojan detection, in spite of this overhead.

Full Text

Authors

References

Citations

Keywords

Related Articles

Footnotes

Back to Top

SECTION III.
Proposed Online Methodology for
HTH Detection

Our proposed methodology core idea is to perform online monitoring of the internal logic values at *probable* HTH insertion sites. Rather than trying to predict what would be the mode of operation of an inserted HTH, we focus on the *expected functionality* of the circuit, and try to detect deviations from the expected sets of logic values at correlated internal nodes.

The primary objectives of our monitoring technique are the following:

- Given a circuit netlist, identify the internal nodes which are the most probable insertion sites for HTHs.
- Select low-overhead checkers at a selected subset of these internal nodes to identify and report logic malfunctions.
- Avoid the need to generate test vectors to propagate the logic value errors to the primary outputs.
- Restrict the number of inserted checkers to keep the hardware overhead within the allowed budget, while achieving maximum HTH detection capability.

The main observation behind our technique is that: *HTH insertion that modifies the logic value at a node disrupts the correlation between the logic value at that node, and the logic values at the nodes in its neighborhood. By “neighborhood” of a node \mathcal{N} which is input to c different gates $\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_c$, we mean the other nodes at the input and output of these c logic gates. Hence, we insert checkers at well-chosen internal locations, termed “important nodes”, to monitor and report logic malfunctions in its vicinity.*

Figure 2 shows an example of the concept, where both single-rail and double-rail checkers have

been shown. For simplicity, first consider the single-rail checker. Suppose, node A in the netlist has been chosen as an important node (the exact algorithm to select such an important node would be described later in this section). We consider the correlation between the logic values at node A , the nodes B and C which are inputs to the same gates that A leads to, and nodes D and E which are the closest nodes to A in its fanout cone. If node A is at a non-controlling value logic-1, it is expected that $D = \sim B$ and $E = \sim C$. On the other hand, if A is at the controlling value logic-0, it is expected that $D = E = 1$. If a HTH is inserted at node A , this correlation would be disturbed. The inserted checker monitors these sets of correlations online, and reports any deviation from the expected behavior through the *error* signal. The dual-rail checker has similar functionality; when no HTH is detected, the output of the checker is “01”, and the checker outputs “10” when a HTH is detected. The checker outputs “00” or “11” (illegal output value combinations) to indicate circuit-level problems with it.



Figure 2.
Example of checker to verify correlation of logic values in the neighborhood of an *important node*. Both single-rail and double-rail checker outputs are shown.

Two-rail checkers are deployed in a system to address the issue of “single point of failure”. If the error indicator signal is itself stuck-at-1 or stuck-at-0, the checker fails to serve its purpose, and might allow error propagation to subsequent functional blocks, which is a highly undesirable situation. Furthermore, if the attacker wires the error indicator point to logic ‘1’ it will stop the entire purpose of checker logic from indicating a malfunction. Hence, usually in two-rail checkers the correct output is dynamic and indicated by “10” or “01”, while error conditions are indicated by “00” and “11” combinations. This property imparts “self-checking” features for the dual-rail checkers, a feature not present in single-rail checkers [34].¹

An inserted HTH can be detected by an applied test vector, if two conditions are simultaneously satisfied: (a) the HTH is triggered, and, (b) the impact of the incorrect logic value caused by the triggered HTH propagates to the primary output of the circuit. So, along with the low trigger probability, the difficulty of propagation of the incorrect logic values at the internal circuit nodes to the primary output also contributes to the low HTH detection coverage without the proposed checkers. The proposed checkers, when inserted, do not affect the trigger condition of the inserted HTH, but avoids the need for the incorrect logic values generated by the triggered HTH to directly propagate all the way to the primary output.

A. Selection of Important Nodes

Simple random selection of multiple nodes would incur unacceptable overhead, while not being satisfactorily effective. Thus, our checker methodology relies on a smart selection of *important nodes*. The algorithm used for this purpose is called *Signal Probability Reliability Analysis* (SPRA) [35]. SPRA is an analytical tool for reliability analysis. SPRA calculates circuit reliability by taking into account logical masking. One positive aspect of using such method is that it is not simulation-based (i.e., it is analytical); thus all possible input scenarios can be taken into account in linear execution time.

SPRA works by taking a gate-level netlist as input and also a list of q values, these being in the $[0,1]$ range. The q values are the individual reliabilities of each node of the circuit. Then, through a smart manipulation of matrices and probabilities that achieves fault propagation, SPRA calculates the overall reliability of the whole circuit. A given gate can influence the circuit reliability negatively by introducing additional error to the signals being propagated through it. A gate can also have a positive influence if it reduces the error probability of a signal or signals. Gates that have a high masking ability (think AND4) can contribute positively. Gates like inverters – that have zero logical masking – only contribute negatively. Gates that have a high fan-out usually contribute negatively too. We refer the reader to [36] for more details of how the SPRA algorithm works.

As a matter of fact, a modified version of SPR was used for our node selection. We are no longer interested in the overall circuit reliability, but instead we want to understand how each gate contributes to that reliability. Algorithm 1 details the required steps for *important node* selection. There is only one input to the node selection procedure: G is a netlist containing all the gates in the circuit, n is the size of G , g is the current target gate and Q is an array of all individual reliability values (the q values). The result from each SPR run is stored in an array called R . Before each SPR run, one and one gate only is selected as the target gate (lines 6-9). The $Q[i]$ value of this target gate is set to zero, meaning that it will ‘corrupt’ all signals that pass through it. All other gates are set the opposite way, i.e., $Q[i] = 1$. The results array from the SPR run R is further manipulated to eliminate all zeros. This is necessary since SPR gives zero as a trivial answer when the target gate is directly connected to a primary output. We assume that an adversary would pick target gates more wisely and therefore do not consider this scenario meaningful. A second manipulation process

Full Text

Authors

References

Citations

Keywords

Related Articles

Footnotes

Back to Top

wisely and therefore do not consider this scenario meaningful. A second manipulation process takes place, so the array R is sorted based on its contents. The elements with the lowest R values are the most important nodes, since they influence the circuit reliability to a great extent.

Algorithm 1

procedure ImportantNodeSelection(G)

$n \leftarrow size(G)$

$g \leftarrow 0$

for $g \leftarrow 0, n - 1$ do

for $i \leftarrow 0, n - 1$ do

if $i = g$ then

$Q[i] \leftarrow 0$

else

$Q[i] \leftarrow 1$

end if

end for

$R_g \leftarrow SPR(G, Q);$

end for

$R \leftarrow eliminateZeros(R)$

$R \leftarrow sort(R)$

end procedure

We consider an attack model whereby the attack causes an immediate functional error on HTH activation, that renders the chip completely useless. This enables minimal number of HTH insertions while having extreme disruption capability, with minimal impact on side-channel parameters such as power consumption. This would in turn, make it difficult to apply side-channel analysis based HTH detection techniques ineffective. In this case, from an adversary's perspective, gates that greatly lower the reliability are good candidates for HTH insertion, which we have considered. On the other hand, some attacks have a late-life target, such that the chip will pass all early test phases and eventually fail in the field later on. This second type of HTH would be inserted by an adversary at gates which mask errors, thereby lowering circuit reliability. Both attack models are possible and likely, We could easily modify our node selection algorithm by sorting the nodes in reverse order, to focus on the second type of HTH. Thus the proposed algorithm is sufficiently generic.

Note that although SPR is an analysis method that is applicable essentially for combinational circuits, the proposed methodology of checker insertion at selected nodes based on SPR can easily be extended to sequential circuits, by performing the analysis of the sequential circuits in “full-scan” mode, whereby the sequential elements are initialized to arbitrary values. However, this straightforward extension does not always lead to very accurate estimations. A relatively complicated but more accurate technique has been proposed in [36].It is a hybrid solution, in which combinational logic is handled by SPRA directly, whereas sequential logic is emulated on a cycle by cycle basis. To each flip-flop, a “reliability counter” is assigned. If, in a given cycle, the flip-flop receives new data from a combinational path, the counter takes the reliability of the incoming signal as its own and updates itself. If, on the other hand, the flip-flop data is not updated and the same value is retained (perhaps due to clock gating), the counter is incremented that represents the chance of failure of the flip-flop itself. These updates to flip-flop counter are interleaved with regular SPRA runs. Interestingly, the counters tend to stabilize after many evaluation cycles, and the reliability of the entire circuit can be calculated by observing the reliability values at the primary outputs.

B. Impact of Checker Insertion

Figure 3 demonstrates the impact of a checker using a small but complete example. Suppose, node

Full Text

Authors

References

Citations

Keywords

Related Articles

Footnotes

Back to Top

$N30$ in the circuit is selected as an “important node”. Now, the inserted HTH is triggered whenever nodes $N10$ and $N20$ are each at logic-0, and the triggered HTH modifies the logic-value at node $N30$. No test vector that has $N5$ at logic-1 can detect any malfunction even if the HTH is triggered, because the NAND gate that generates the output value $Nout2$ masks the impact of the triggered HTH. However, with an inserted HTH, the checker easily detects the incorrect logic values at the internal node(s).




Figure 3

Figure 3.

Impact of checker on inserted HTH detectability, with node $N30$ as an “important node”. The triggered HTH cannot be detected for any input test vector when $N4 = 0$, but the checker can detect such a triggered HTH.

C. Resistance to Reverse-Engineering Based Attacks

One of the major advantages of the proposed approach is that the checkers do not have a specific fixed structure. Each inserted checker is designed in a custom way after considering the neighborhood logic of an internal important node, which helps it to mingle easily with the surrounding logic gates, thus making their identification for a cell-replacement attack difficult. In addition, the proposed technique can be combined with self-authentication techniques such as BISA [27] to resist cell-replacement or cell-tampering types of attacks.

D. System Architecture for Design-for-Trust

The error signals generated by the fault checkers distributed throughout the IC are combined using a final checker circuit. As described before, security and fault tolerance are important prerequisites for building reliable systems with enhanced levels of trust. From the overall system perspective it is difficult to distinguish random and unintentional hardware faults from malicious faults caused by HTH activation. In the nano-scale era, it will become increasingly difficult and expensive to manufacture/maintain totally defect-free or fault-free integrated circuits. Therefore, in the final checker we propose to use totally self checking circuits. Self-checking circuits, in general, are that class of circuits in which occurrence of a fault can be determined by observation of the outputs of the circuits. An important subclass of these self-checking circuits is known as *Totally Self-checking* (TSC) circuits [37]. A circuit is TSC if a fault in the circuit cannot cause an error in the outputs without detection of the fault. Figure 4 shows the general chip-level architecture for reliability enhancement against HTH insertion. We assume totally self-checking comparators [38] at each of the checker to verify correlation of logic values in the neighborhood of an important node.

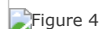


Figure 4

Figure 4.

Chip-level architecture for HTH detection.

The two-rail checker outputs are finally combined to form a two-rail output signal. These self-checking comparators generate two-rail output signals, with values depending on the Trojan detection condition (as described in Figure 2), which are combined as shown in Figure 4. The final checker will produce a two-railed checker output (f and g). If we use two rail checker, the input to the final checker will be k -out-of- $2k$ code. Efficient implementation self-checking checkers are available in the literature [39]. When no HTH is detected, the output of the final checker will be “01”, and the checker outputs “10” when a Trojan is detected. The checker outputs “00” or “11” to indicate problem with one of the checkers, or in the final checker. In Figure 4, the different blocks can be different hardware IP cores included in a system-on-chip (SoC). The error signals can be routed over a SoC using standard on-chip interconnect infrastructure available on modern SoCs, e.g., *Advanced Microcontroller Bus Architecture* (AMBA) [40]. This chip-level architecture can be expanded similarly to cover a system-level HTH detection infrastructure, and based on the system-level detection signal, proper countermeasures might be activated.

E. Automated Design Flow

The complete automated design flow is demonstrated in Figure 5. Given a circuit netlist, its internal nodes are first sorted according to their level of importance following the algorithm described above. Then, given a threshold (upper bound) A_{th} on the area overhead, a trial N_c number of TSC checkers with the appropriate logic are inserted in the netlist, the checker outputs are combined in a final TSC checker, and the modified netlist is re-synthesized. A_{th} was set empirically so as to have sufficient Trojan detection coverage, while N_c was obtained by dividing the target area overhead amount for the benchmark circuit by the area of the synthesized two-rail checker shown in Figure 2. If the area overhead of the resynthesized design is found to be within the threshold, the modifications are accepted. Otherwise, the number of modifications N_c is reduced by one, and the

Full Text

Authors

References

Citations

Keywords

Related Articles

Footnotes

Back to Top

process continued until the overhead is within the allowable threshold.



Figure 5.
Flow chart of the proposed DFTr methodology for HTH detection.

SECTION IV.
Experimental Result and Discussions

A. Experimental Setup

The proposed technique was applied on the generic Verilog netlists of a subset of the ISCAS-85 combinational circuits [41]. The original netlists and the modified netlists were synthesized using *Synopsys Design Compiler*, using a 45 nm standard cell library. Exhaustive test patterns were generated for the smallest benchmark circuit (c17) considered by us, and random test patterns were used for the rest of the benchmark circuits. The entire automated design methodology was implemented in C++.

The HTHs we considered were small combinational Trojans of the form shown in Figure 1(a), with a maximum of four trigger nodes. The HTH selection methodology followed the technique described in [6]. Thus, the HTH instances were selected to have low probability input trigger combinations, and were verified to be “true Trojans”, i.e., the Trojans were triggerable and, upon triggering, the logic malfunction caused by them propagates to the primary output. We considered two scenarios of HTH insertion:

- The HTH instances were inserted within a radius of two logic gates of the inserted checker. This case represents an optimistic scenario, since an intelligent adversary might recognize the checker insertion locations by analysis of the netlist obtained by reverse engineering, and intentionally avoid the proximity of the checker for actual Trojan insertion. Hence, the Trojan detection coverage obtained in this scheme can be considered to represent an “upper bound” of the effectiveness of the proposed scheme.
- The HTH instances were inserted at a radius of ten or more logic gates away from the inserted checker. This is the more probable scenario with respect to Trojan insertion.

B. HTH Detection Coverage

We considered the number of checker insertion locations to be such that the target area overhead of the modified design, including the final checker, is about 10 percent than the original design (except the three smallest benchmark circuits). Table 1 shows the improvement in Trojan detection coverage for Trojans inserted close to the checker locations, while Table 2 shows the improvement in Trojan detection coverage for Trojans inserted away from the checker locations. The experimental results clearly demonstrate the effectiveness of the proposed technique, with substantial improvement in detection coverage. As expected, the improvements are more pronounced for Trojans inserted near the checkers. For all the benchmark circuits considered (other than c17 for which we had already presented results for exhaustive testset), application of 10X the number of test vectors reported in Tables 1 and 2, failed to reach even 30 percent of the HTH detection coverage achieved with checker insertion. Randomly generated test vectors are usually unable to detect well-designed HTHs (especially those designed to be activated by the simultaneous triggering of low signal probability nodes), which has also been reported in previous works [6].

TABLE 1. HTH detection coverage improvement by HTH insertion near checker.



TABLE 2. HTH detection coverage improvement by HTH insertion away from checker.



C. Design Overhead

Table 3 presents the area, delay and power overheads of the modified designs with respect to the original designs, along with the target area overhead threshold (A_{th}) and the trial number of

[Full Text](#)

[Authors](#)

[References](#)

[Citations](#)

[Keywords](#)

[Related Articles](#)

[Footnotes](#)

[Back to Top](#)

checkers to be inserted (N_c). It should be noted that the difference between the target number of checkers and the actual number of checkers inserted is not more than 2, for all the benchmark circuits considered. This keeps the design effort within acceptable levels, by limiting the number of resynthesis runs required. The test vectors were applied at intervals of 10 ns. Note that the overhead percentages of the three smallest ISCAS-85 circuits considered are relatively large, because of the small sizes of these circuits, and hence these overheads should not be considered representative.

TABLE 3. Design overheads of proposed DFTr technique for HTH detection.

| |
|--|
|  Table 3. |
|--|

D. Recent Trends in System Security

The severe security implications introduced by modern design and offshore manufacturing practices has led researchers to explore alternative security methodologies. Instead of depending only on hardware monitoring approaches, a hardware-software co-design approach to ensure security has been proposed [42], [43]. Also, novel nano-devices such as Memristors and *Spin Torque Transfer RAM* (STTRAM) have unique properties which makes it possible to design and implement low-overhead hardware security primitives [44] , [45]. These devices often allow integration with standard CMOS circuitry, which makes them even more attractive for system integration. The techniques proposed in this paper, if implemented using a hardware-software co-design approach or implemented using nano-devices, would help to decrease the area, power and performance overheads. This would, in turn, lead to higher HTH detection coverage, at a given area overhead, than the results presented in this paper.

SECTION V.
Conclusion

We have developed a low-overhead online checking technique for HTH detection at gate level, and its application to develop a system architecture directed towards design-for-trust. We have automated the design methodology, and our experimental results demonstrate the effectiveness of the proposed technique, with close to 100 percent HTH detection coverage at 10 percent area overhead and acceptable design effort overhead for most of the benchmark circuits considered. The technique avoids the need for unconventional manufacturing flows such as split manufacturing, difficulty of key management in obfuscation based schemes, is resistant to reverse-engineering based attacks such as “cell-replacement attack”, and can be extended easily for to secure system design. Our future research would be directed towards integrating the proposed methodology with fault-tolerant design techniques for emerging design nano-fabrics with high device defect density.

ACKNOWLEDGMENTS

This research was made possible by a Visiting Fellowship awarded by Royal Academy of Engineering (U.K.) to Rajat Subhra Chakraborty (2014-2015).

Full Text

Authors

References

Citations

Keywords

Related Articles

Footnotes

Back to Top

| | |
|------------------|---|
| Authors | ▼ |
| References | ▼ |
| Citations | ▼ |
| Keywords | ▼ |
| Related Articles | ▼ |
| Footnotes | ▼ |

IEEE Account

» Change Username/Password

» Update Address

Purchase Details

» Payment Options

» Order History

» View Purchased Documents

Profile Information

» Communications Preferences

» Profession and Education

» Technical Interests

Need Help?

» **US & Canada:** +1 800 678 4333

» **Worldwide:** +1 732 981 0060

» Contact & Support

Full Text

Authors

References

Citations

Keywords

Related Articles

Footnotes

Back to Top

| About IEEE *Xplore* | Contact Us | Help | Accessibility | Terms of Use | Nondiscrimination Policy | Sitemap | Privacy & Opting Out of Cookies

A not-for-profit organization, IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.
© Copyright 2018 IEEE - All rights reserved. Use of this web site signifies your agreement to the terms and conditions.