ANALYZING CIRCUIT VULNERABILITY TO HARDWARE TROJAN INSERTION AT THE BEHAVIORAL LEVEL

Hassan Salmani and Mohammed Tehranipoor
Electrical and Computer Engineering, University of Connecticut
{salmani_h,tehrani}@engr.uconn.edu

Abstract — **Considerable attention has been paid to hardware Trojan detection and prevention. However, there is no existing systematic approach to investigate circuit vulnerability to hardware Trojan insertion during development. We present such an approach to investigate circuit vulnerability to Trojan insertion at the behavioral level. This novel vulnerability analysis determines a circuit's susceptibility to Trojan insertion based on statement hardness analysis as well as observability of circuit signals. Further, the Trojan detectability metric is introduced to quantitatively compare the detectability of behavioral Trojans inserted into different circuits. This creates a fair comparison for analyzing the strengths and weaknesses of Trojan detection techniques as well as helping verify trustworthiness of a third party Intellectual Property (IP).**

I. INTRODUCTION

Integrated Circuits (ICs) are at the core of any electronic system today, and their security grounds the security of entire system. Notwithstanding the central impact of ICs security, the horizontal IC design process, where each step may be performed by various parties at different locations, is widespread throughout the industry [1] . Realized by modifying design implementation, hardware Trojans can interfere with any step of the design process. A hardware Trojan may undermine a system's confidentiality by leaking secret information or can abrogate system availability by performing a malfunction.

Circuits are vulnerable to Trojan insertion in every stage of their development due to globalization. Third-party intellectual properties (IPs) are used extensively in high-level implementation. Their details are not delivered most of the time; therefore, IPs may stealthily deviate from specifications determined by designers and system integrators. In an off-shore design house, a rogue designer can tamper with synthesized circuits and insert extra gates. Circuit masks can also be manipulated in foundries to change circuit characteristics. Consequently, analyzing a circuit's vulnerability to Trojan insertion at different levels is a key step towards trusted design development [2].

Hardware Trojans have negligible effect on a circuit and rarely become fully activated. Several techniques have been suggested to prevent Trojan insertion or to facilitate their detection. To enhance Trojan detection resolution, some techniques have proposed embedding monitoring systems into main circuits to capture any abnormality in circuit performance or power consumption [3][4][5][6]. Several techniques recommended design-for-hardware-trust aimed to magnify Trojan impact during authentication [7][8][9]. Further, several side-channel-based techniques have been introduced to distinguish the contribution of Trojans from the impact of process variations in a circuit [10][11][12][13]. Hardware Trojan design and implementation were presented in [14][15][16][17].

THIS WORK WAS SUPPORTED IN PART BY THE NATIONAL SCIENCE FOUNDATION (NSF) UNDER GRANTS CNS-0844995 AND CNS-1059390.

Although there has been a significant amount of work on hardware Trojan detection and prevention, no systematic approach to assess the susceptibility of a circuit to Trojan insertion has been developed. Sections in a circuit with low controllability and observability are considered potential areas for implementing Trojans. This necessitates a thorough circuit analysis to identify potential Trojan locations. Furthermore, there is little or no work on developing metrics to determine difficulty of detecting a hardware Trojan. To address the above issues, in this paper, we present a novel and comprehensive vulnerability analysis at the behavioral level, as behavioral-level IPs are commonly used in modern systems-on-chips (SoCs). The analysis quantifies the difficulty of activating each line of a code and observing internal signals and primary inputs through primary outputs. In the following, we determine Trojan detectability based on their activation and observability. The Trojan detectability offers a fair comparison of different detection techniques. The proposed analysis is a vector-less approach and does not require testbench whose development by itself is a challenging task. Furthermore, the execution time complexity of this analysis has a liner relation with circuit size that guarantees its low execution time.

In the following, Section II describes design vulnerability to Trojan insertion at the behavioral level. Simulation results are presented in Section III. Finally, Section IV concludes the paper.

II. VULNERABILITY ANALYSIS AT THE BEHAVIORAL LEVEL

At the behavioral level, a hardware description language (HDL), such as VHDL or Verilog, is used to describe a circuit by concurrent algorithms (behavioral). Each algorithm itself is sequential meaning it consists of a set of statements that are executed one after another. Figure 1 shows our proposed circuit vulnerability analysis flow to Trojan insertion at the behavioral level. The data and control flows of the circuit determine the difficulty of executing a statement and the observability of internal signals. *Statement Analysis* in Figure 1 measures statement weight and statement hardness of a statement which quantify conditions under which the statement executes. In the following, *Observerability Analysis* evaluates reachability of signals from each other and determines their observability through circuit primary outputs. A circuit at the behavioral level is vulnerable to Trojan insertion where statement hardness is high or observability is low. A Trojan at the behavioral level (also called "a behavioral Trojan") can change a statement rarely executed or carry out an attack through a signal with very low observability. *Trojan Detectability Analysis* gauges the detection of a malicious change of a statement or the inclusion of a new statement based on its weight and observability of its target signal.

II-A. Statement Analysis

The proposed vulnerability analysis at the behavioral level quantifies (i) statement hardness for each circuit statement and (ii) observability for each circuit signal, based on a weighted value range. Adopted from the work presented in [18] to improve

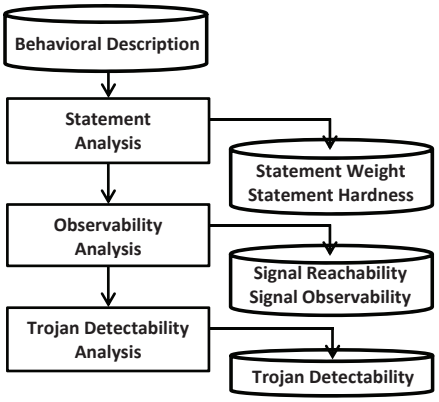

Fig. 1. Vulnerability analysis flow at the behavioral level.

the accuracy of static value and branch prediction in compilers, the notation $\{W[L,U]\}$ is developed for each signal where W represents the weight of the value range, and L and U show the lower and upper limits of the value range. Circuit control and data flows determine W, L and U for each statement. The technique creates a condition stack to track the circuit control flow. Further, it generates an individual stack for each signal to pursue the circuit data flow. While the circuit code is being parsed, observing a condition block limiting the value range of a signal, such as loop and condition statements, pushes a new condition into the condition stack and a new weighted value range into the stack of the signal. Reciprocally, exiting a condition block pops from the condition stack and may update the stacks of signals.

A weighted value range is determined by a condition or assignment statement. The weight of a range is defined as $\left(\frac{U-L+1}{U_O-L_O+1}\right)$, where U and L are the upper and lower limits of the controlling signal at the top of the condition stack, and U_O and L_O are the declared upper and lower limits of the controlling signal. If different assignment statements to one target signal take place under different conditions in one condition block, the weighted value ranges of the target signal will merge when exiting the condition block. Further, the statement hardness is defined as the reverse of the weight of the controlling signal of that statement (*the statement weight*).

The technique was applied to a small behavioral code, shown in Figure 2, and the statement hardness and the weighted value range of controlling signals X, K, and P are presented in Table I. Further, Figure 3 shows the condition stack and stacks for the controlling signals over the code execution.

The program in Figure 2 consists of one sequential block between Lines 13 and 32. In Line 13, where the block begins, the condition stack is empty. The controlling signals are set to their full range with the weight of 1. The first statement in Line 14 executes unconditionally and its statement hardness is set to 1. The loop in Line 14 limits the controlling signal X between 0 and 9, so $L = 0$ and $U = 9$. While L_O = 0 and $U_O = 15$, as declared in Line 5, the weight of the value range is $(\frac{9-0+1}{15-0+1} =)0.625$. Therefore, the hardness of the statement in Line 15 is $\frac{1}{0.625}$. Represented as X_{14} in the condition stack, where the subscripted 14 shows the line number, the signal X is pushed to the condition stack in Line 14 and $\{0.625[0,9]\}$ into the stack of signal X.

The immediate condition statement in the next line, Line 15, again pushes the signal X (X_{15}) into the condition stack and pushes the new weighted value range $\{0.125[0,1]\}$ into the stack of signal X. The hardness of Line 16 is $\frac{1}{0.125}$ and the assignment statement for this line defines a new weighted value range for the signal P; the weighted value range, $\{0.125[0,1]\}$, is then pushed

```
1.  LIBRARY IEEE;
2.  USE IEEE.STD_LOGIC_1164.ALL;
3.  ENTITY EXAMPLE IS
4.  PORT( CLK : IN BIT;
5.          X : IN INTEGER RANGE 15 DOWNTO 0;
6.          K : IN INTEGER RANGE 15 DOWNTO 0;
7.          Z : OUT INTEGER RANGE 15 DOWNTO 0);
8.  END ENTITY EXAMPLE;
9.  ARCHITECTURE SIMPLE OF EXAMPLE IS
10. BEGIN
11. PROC1: PROCESS (CLK)
12. VARIABLE P : INTEGER RANGE 15 DOWNTO 0;
13. BEGIN
14.         FOR X IN 0 TO 9 LOOP
15.                 IF ( X < 2 ) THEN
16.                         P := 1 - X;
17.                 ELSIF ( X > 5 ) THEN
18.                         IF ( K = 7) THEN
19.                                 P := 2 * X;
20.                         ELSE
21.                                 P := 2 + K;
22.                         END IF;
23.                 END IF;
24.         END LOOP;
25.         IF ( P < 7 ) THEN
26.                 IF ( X < 7 ) THEN
27.                         IF ( P > 2 ) THEN
28.                                 Z <= P;
29.                         END IF;
30.                 END IF;
31.         END IF;
32. END PROCESS PROC1;
33. END ARCHITECTURE SIMPLE;
```

Fig. 2. A sample behavioral code.

into the stack of signal P. The ELSE statement in Line 17 in Figure 2 pops from the condition stack (X_{15}) and from the stack of signal X. Afterwards, the top element of the condition stack is X_{14}, so the statement hardness of Line 17 is $\frac{1}{0.625}$. Further, the ELSE statement in Line 17 pushes X (X_{17}) into the condition stack, generates a new weighted value range $\{0.25[6,9]\}$, and pushes into the stack of signal X.

In the following, the hardness of executing statements in Line 18 is $\frac{1}{0.25}$. In Line 18, a new condition block is defined over K inside the condition block already defined in Line 17. The last condition statement limits the controlling signal K to 7, and the new weighted value range of K, i.e. $\{0.25[7,7]\}$ is pushed into the stack of signal K and K_{18} into the condition stack. By executing the assignment statement in Line 19, a new weighted value range $\{0.015625[12,15]\}$ is pushed into the stack of signal P. The ELSE condition statement in Line 20 pops K_{18} from the condition stack and removes the top range of the stack of signal K. Afterwards, the top controlling signal is X_{17} that determines the statement hardness of Line 20. Then, the new value range of the signal K, $\{0.25[0,6], 0.25[8,15]\}$ is pushed into the stack of signal K. Finally, the K_{20} is pushed into the condition stack and determines the statement hardness of Line 21.

The assignment statement in Line 21 determines new value ranges for the signal P, i.e., $\{0.234375[2,8], 0.234375[10,15]\}$. As P was the target of two assignments in Lines 19 and 21 in one condition block, the two top weighted value ranges of P are merged and saved as one element $\{0.25[2,8], 0.25[10,15]\}$ at the top of the stack of signal P. The condition stack is popped two times with END IF in Lines 22 and 23. In Line 23 the last weighted value range of P is merged with the weighted value range of P, defined in Line 16, as they are in the same condition block. The new weighted value range $\{0.625[0,8], 0.625[10,15]\}$ is pushed into the stack of signal P. The END LOOP in Line 24 pops from the condition stack, and it becomes empty. Line 25 executes unconditionally, defines a new weighted value range $\{0.4375[0,6]\}$ for the signal P, and pushes P_{25} into the condition stack. The condition statement in Line 26 defines $\{0.19140625[0,6]\}$ as the new weighted value range of the signal X and pushes X_{26} into the condition stack.

With the next line, a new weighted value range for the controlling signal P , i.e. $\{0.0478515625[3,6]\}$, is defined, and

Table I. Statement hardness and value range of controlling signals for the code in Fig. 2.

Line number	Statement hardness ($1/T_W$)	Value range		
		X	K	P
13	1	$\{1[0,15]\}$	$\{1[0,15]\}$	$\{1[0,15]\}$
14	1	$\{1[0,15]\}$	$\{1[0,15]\}$	$\{1[0,15]\}$
15	1/0.625	$\{0.625[0,9]\}$	$\{1[0,15]\}$	$\{1[0,15]\}$
16	1/0.125	$\{0.125[0,1]\}$	$\{1[0,15]\}$	$\{0.125[0,1]\}$
17	1/0.625	$\{0.625[0,9]\}$	$\{1[0,15]\}$	$\{0.125[0,1]\}$
18	1/0.25	$\{0.25[6,9]\}$	$\{1[0,15]\}$	$\{0.125[0,1]\}$
19	1/0.015625	$\{0.25[6,9]\}$	$\{0.25[7,7]\}$	$\{0.015625[12,15]\}$
20	1/0.25	$\{0.25[6,9]\}$	$\{1[0,15]\}$	$\{0.015625[12,15]\}$
21	1/0.234375	$\{0.25[6,9]\}$	$\{0.25[0,6], 0.25[8,15]\}$	$\{0.234375[2,8], 0.234375[10,15]\}$
22	1/0.25	$\{0.25[6,9]\}$	$\{1[0,15]\}$	$\{0.25[2,8], 0.25[10,15]\}$
23	1/0.625	$\{0.625[0,9]\}$	$\{1[0,15]\}$	$\{0.625[0,8], 0.625[10,15]\}$
24	1	$\{1[0,15]\}$	$\{1[0,15]\}$	$\{0.625[0,8], 0.625[10,15]\}$
25	1	$\{1[0,15]\}$	$\{1[0,15]\}$	$\{0.625[0,8], 0.625[10,15]\}$
26	1/0.4375	$\{1[0,15]\}$	$\{1[0,15]\}$	$\{0.4375[0,6]\}$
27	1/0.19140625	$\{0.19140625[0,6]\}$	$\{1[0,15]\}$	$\{0.4375[0,6]\}$
28	1/0.0478515625	$\{0.19140625[0,6]\}$	$\{1[0,15]\}$	$\{0.0478515625[3,6]\}$
29	1/0.19140625	$\{0.19140625[0,6]\}$	$\{1[0,15]\}$	$\{0.4375[0,6]\}$
30	1/0.4375	$\{1[0,15]\}$	$\{1[0,15]\}$	$\{0.4375[0,6]\}$
31	1	$\{1[0,15]\}$	$\{1[0,15]\}$	$\{0.625[0,8], 0.625[10,15]\}$
32	1	$\{1[0,15]\}$	$\{1[0,15]\}$	$\{0.625[0,8], 0.625[10,15]\}$

			K_{18}	K_{20}						P_{27}			
	X_{15}	X_{17}	X_{17}	X_{17}	X_{17}				X_{26}	X_{26}	X_{26}		
X_{14}	X_{14}	X_{14}	X_{14}	X_{14}	X_{14}	X_{14}		P_{25}	P_{25}	P_{25}	P_{25}	P_{25}	
Line 14	Line 15	Line 17	Line 18	Line 20	Line 22	Line 23	Line 24	Line 25	Line 26	Line 27	Line 29	Line 30	Line 31

Fig. 3. The control flow for the code in Fig. 2.

Table II. The value range of internal and output signals for the code in Fig. 2.

Signal name	Value range
P	$\{0.625[0,8], 0.625[10,15]\}$
Z	$\{0.0478515625[3,6]\}$

P_{27} is also pushed into the condition stack. The assignment statement in Line 28 assigns the signal P to the output signal Z with the statement hardness of $\frac{1}{0.0478515625}$. The next three lines pops P_{27}, X_{26}, and P_{25} from the conditional stack, and Lines 31 and 32 are executed unconditionally.

A signal's weighted value range indicates the range of values the signal can take. Constrained by circuit data and control flows, the range may only cover part of the original declared range of the signal. The weight of the range, determined by statement hardness, represents the difficulty of assigning the range to the signal. Table II shows the weighted value range of the output signal Z and the internal signal P for the code in Figure 2.

The declared value range of P and Z are $[0, 15]$. However, the value range of the signal P considering the circuit control and data flows does not include the value 9. Similarly, Table II shows that the output signal Z only covers the value range of $[3, 6]$ with comparatively low weight, and the signal does not cover 75% of its declared range. Uncovered value ranges can be potentially used for behavioral Trojans implementation because circuit verification and the normal operation of the circuit do not reach values out of range. Knowing that, an attacker may design a Trojan to propagate unexpected values and initiate a malfunction.

Independent from circuit input vectors, statement hardness provides a quantitative measure of the difficulty of executing a statement. A statement with a high level of hardness is rarely executed and its correctness cannot be fully examined with applying a limited number of test input vectors in a testbench form. In the example above, the assignment statement in Line 19 with a statement hardness of (1/0.015625=)64 is the most difficult statement in the code to be exercised due to nested conditional statements which control the execution of the statement. Therefore, modifications to the statement or the inclusion of new statements under the same condition will be hard to detect. This analysis reveals parts of a circuit that are more vulnerable to Trojan insertion. As another measure, the observability of a signal characterizes the difficulty of observing the signal through a circuit primary output.

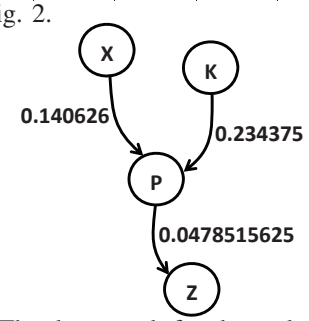

Fig. 4. The data graph for the code in Fig. 2.

II-B. Observerability Analysis

A behavioral-level Trojan usually targets a signal with low observability to carry out an attack. The vulnerability analysis technique develops a weighted data graph of a circuit while it parses circuit code in order to calculate observability along with statement hardness. The graph shows the connectivity of signals that appeared in code . Figure 4 shows the data graph of the code in Figure 2.

The nodes of the data graph are circuit signals, and directed edges show their dependency. Edges are weighted with the sum of the weights of assignment statements that represent the source node as an input and the destination node as the target signal. For example, the signal X appears in two assignment statements in Lines 16 and 19 where the target signal is the signal P in both. The weights of the assignment statements, as shown in Table I, are 0.125 and 0.015625, respectively; therefore, the weight of the

Table III. The reachability and observability of signals for the code in Fig. 2.

Signal name	Target signal	Reachability	Observability(T_O)
X	Z	0.006729	0.006729
X	P	0.140625	0
P	Z	0.047851	0.047851
K	Z	0.011215	0.011215
K	P	0.234375	0

edge from the signal X to the signal P is $(0.125 + 0.015625 =)0.140625$. In another case, the weight of the edge from the signal K to the signal P is the weight of the assignment statement in Line 21, which is 0.234375, as shown in Table I. Based on the data graph, it is possible to calculate the reachability of signals from each other and the observability of signals at circuit outputs. Table III shows the reachability and observability of signals in the circuit in Figure 2, based on the data graph in Figure 4.

The vulnerability analysis flow at the behavioral level determines the reachability of signals that can directly or indirectly reach each other. With the assignment statement in Line 28, the signal P directly reaches the signal Z with a reachability of 0.047851, which is the weight of edge from P to Z. As the signal Z is an output signal, the observability of H is also 0.047851.

The signal X reaches the signal P, and its reachability is 0.140625. But its observability is 0 as the signal P is not an output signal. However, the signal X is observable through the signal Z, an output signal. The reachability of the signal X to the signal Z is $(0.140625 \times 0.047851 =)0.006729$, and with the same value, it is observable. In a similar manner, the reachability and observability of signal pairs can be calculated. The signal P has the highest observability, as it is directly assigned to the output signal Z. On the other hand, the signal X is only observable with the minimum value of 0.006729 thru the output signal Z.

Reachability of a signal pair signifies driving strength of a source signal in determining the final value of a destination signal. As shown in Figure 4, the signal P is driven by the signals X and K. The reachability of the pair K and P is 0.234375 and that of the pair X and P is 0.140626, that is, the signal K has more contribution in the final value of the signal P. Confirmedly Table I indicates the signal K determines the widest range of the signal P with a higher weight compared with the signal X which derives the signal P in Lines 16 and 19 with smaller ranges and lower weights.

The vulnerability analysis flow determines the statement hardness of each statement in a code. The statement hardness exposes parts of the code which are more vulnerable to Trojan insertion. Further, it is possible to obtain the weighted value range of circuit signals at any line of the code. This information can be used to determine the root of high statement hardness. The flow makes it also possible to calculate the reachability of signals from each other and their observability by an output signal. The reachability analysis determines to what extent a signal can impact a target signal, and the observability analysis indicates how difficult it is to monitor an internal signal by an output signal.

The vulnerability analysis flow consists of two main steps. In the first step, as a code is being parsed, the hardness of each statement is calculated and a data graph is developed. In the following step, the reachability and observability of signals are calculated using the data graph. The complexity of the first step is $O(L)$, where L is the number of statements. Suppose S is the number of signals; the complexity of the second step would be $O(S \times S)$. Therefore, the complexity of the vulnerability analysis algorithm is $O(L) + O(S^2)$. It is expected that the number of signals in a circuit is much less than the number of statements ($S \ll L$), thus the complexity of the vulnerability analysis algorithm is approximated to $O(L)$.

```
1.  LIBRARY IEEE;
2.  USE IEEE.STD_LOGIC_1164.ALL;
3.  ENTITY EXAMPLE IS
4.  PORT(CLK : IN BIT;
5.          X : IN INTEGER RANGE 15 DOWNTO 0;
6.          K : IN INTEGER RANGE 15 DOWNTO 0;
7.          Z : OUT INTEGER RANGE 15 DOWNTO 0);
8.  END ENTITY EXAMPLE;
9.  ARCHITECTURE SIMPLE OF EXAMPLE IS
10. BEGIN
11. PROC1: PROCESS (CLK)
12. VARIABLE P : INTEGER RANGE 15 DOWNTO 0;
13. BEGIN
14.         FOR X IN 0 TO 9 LOOP
15.                 IF ( X < 2 ) THEN
16.                         P := 1 + X;          --Trojan 1(TjB-Loc1)
17.                 ELSIF ( X > 5 ) THEN
18.                         IF ( K = 7) THEN
19.                                 P := 2 + X;  --Trojan 2 (TjB-Loc2)
20.                         ELSE
21.                                 P := 2 + K;
22.                         END IF;
23.                 END IF;
24.         END LOOP;
25.         IF ( P < 7 ) THEN
26.                 IF ( X < 7 ) THEN
27.                         IF ( P > 2 ) THEN
28.                                 Z <= P - 1;  --Trojan 3 (TjB-Loc3)
29.                         END IF;
30.                 END IF;
31.         END IF;
32. END PROCESS PROC1;
33. END ARCHITECTURE SIMPLE;
```

Fig. 5. A behavioral-Trojan inserted code for the code in Fig. 2.

Table IV. Trojan detectability at the behavioral level for the code in Fig. 5.

Trojan	T_W	T_O	$T_{Detectability}$
TjB-Loc1	0.125	0.047851	0.005981
TjB-Loc2	0.015625	0.047851	0.000747
TjB-Loc3	0.047851	1	0.047851

II-C. Trojan Detectability Analysis

Behavioral Trojans may target signals with low observability and lay where statement hardness is high to reduce their detection probability. To compare the difficulty of detecting behavioral Trojans across different circuits, here, we define behavioral Trojan detectability based on statement weight and observability measures. For each Trojan statement, statement weight (T_W) and the observability of its target signal (T_O) are determined, and the Trojan statement detectability is then defined as $T_{Detectability} = T_W \times T_O$.

For example, three Trojans are inserted in the code presented in Figure 2 by changing assignment statements, as shown in Figure 5. Table IV shows their detectability. The first behavioral Trojan (TjB-Loc1, where B represents "behavioral level") is realized by manipulating the statement in Line 16 and changing the subtraction operator to the addition operator. The weight of statement (T_W) is 0.125, according to Table I, and the observability of the target signal (T_O), the signal P, is 0.047851, based on Table III. Therefore, the detectability of Trojan TjB-Loc1 is $TjB - Loc1_{Detectability} = T_W \times T_O = 0.125 \times 0.047851 = 0.005981$. By modifying the assignment statement in Line 19, the second behavioral Trojan (TjB-Loc2) is implemented. The statement weight and the observability of the target signal are 0.015625 and 0.047851, respectively; therefore, $TjB - Loc2_{Detectability} = T_W \times T_O = 0.015625 \times 0.047851 = 0.000747$. The last behavioral Trojan (TjB-Loc3) is carried out by changing the assignment statement in Line 28. The detectability of TjB-Loc3 is $TjB - Loc3_{Detectability} = T_W \times T_O = 0.047851 \times 1 = 0.047851$.

The lower a Trojan detectability ($T_{Detectability}$) is, the harder the Trojan is to be detected. I n the above example, TjB-Loc2 is the hardest Trojan to detect with the lowest detectability ranking of 0.000747 because its statement weight is the minimum and the target signal P has low observability. Among TjB-Loc1 and TjB-Loc3, the latter is directly observable through the output

Table V. Benchmarks characteristics.

Name	# VHDL Lines	PI	PO	Function
b01	115	4	2	FSM
b02	55	3	1	IsBCD
b03	175	6	4	Arbiter
b04	93	13	8	MinMax
b05	278	3	6	Memory contents elaboration

signal Z, and its statement weight is higher. Hence, TjB-Loc3 has higher detectability, and TjB-Loc1 is ranked second . Note that other potentially inserted Trojans (inserting new statements, loops, etc) in a behavioral HDL code can be evaluated in a similar manner.

III. SIMULATION RESULTS

The proposed vulnerability analysis is applied to several ITC99's benchmarks [19], and in this section results on b01, b02, b03, b04, and b05 benchmarks are presented. Table V demonstrates the details of each benchmark. As there is no unique way of implementing a circuit at a high level of abstraction (i.e. , here, behavioral level), we modified the circuits slightly to prepare for analysis using our automated flow, while keeping the original functionality intact.

b01 circuit is a finite state machine which compares serial flows and has four primary inputs and two primary outputs. With three inputs and one output, b02 recognizes BCD numbers. b03 is an arbiter, accepting four lines of request and issuing a grant for one of them. Having two bus lines and three input controlling signals, b04 computes the minimum and maximum over an input stream. Finally, b05 elaborates the contents of a memory and has three inputs and six outputs. Except b05 circuit, the circuits are sequential and consist of only one VHDL process statement. Each process statement has one condition block. b05 has three process statements of which one consists of a state machine, and the others are composed of nested condition blocks.

The vulnerability analysis for the benchmarks are presented in Table VI. The minimum statement hardness of the five benchmarks is 1, which means the corresponding statement is executed unconditionally. Nested conditional statements increase statement hardness. The highest statement hardness of b01 is (1/0.015625 =) 64, where the corresponding statement is executed when four conditions are met. b01 has one internal signal which is not observable through any of its outputs; therefore, the minimum observability for b01 is 0. Two primary inputs are observable through one of its outputs and determine the maximum observability of b01, which is 0.5.

b02 benchmark is the smallest circuit, and its minimum and maximum statement hardnesses are 1 and (1/0.035714286 =) 28, respectively. The only output of the circuit is assigned constant values, so no internal signals or primary inputs are observable. Hence, the minimum and maximum observability for b02 are 0.

None of primary inputs of b03 is observable through its primary outputs; therefore, the minimum observability is 0. The only internal signal that is assigned constant values is observable with the maximum observability of 0.1666. The maximum statement hardness of b03 is (1/0.005208333 =) 192, and the corresponding statement is executed when four conditions are met.

All internal signals of b04 circuit are observable only through its primary output, and b04 has the highest maximum observability with value of 0.25. The minimum observability is 0, which belongs to some primary inputs. The maximum statement hardness of b04 is (1/0.041666667 =) 24, whose corresponding statement is executed when four conditions are met.

b05 is the largest circuit and has the highest maximum statement hardness among all with the value of (1/2.26E-06 =) 442477. The considerable high statement hardness is attributed to the deep nested condition blocks. The corresponding statements are executed when 11 conditions are met. The minimum and maximum observability of b05 are 0 as all outputs are assigned to constant values.

Table VI. Vulnerability analysis for the benchmarks.

Name	Statement hardness		Observability		CPU runtime (sec)
	Min	Max	Min	Max	
b01	1	1/0.015625	0	0.5	0.02
b02	1	1/0.035714286	0	0	0.0099
b03	1	1/0.005208333	0	0.166	0.04
b04	1	1/0.041666667	0	0.25	0.41
b05	1	1/2.26E-06	0	0	0.14

The vulnerability analysis is performed on a machine equipped with Intel Xeon @ 2.66GHz processor, 6MB cache and 23.57GB memory. The reported CPU runtime in Table VI indicates the execution of the flow in a small fraction of a second for all benchmarks. Figure 6 shows the distribution of statement weight for benchmarks b01 through b05. "Normalized Frequency" in the figure indicates the number of lines with a specific statement weight in a benchmark divided by the number of VHDL lines inside the benchmark's VHDL process statements. As soon, a portion of statements in the circuit presents very low statement weight. Assuming that any statement with statement weight lower than 0.0001 is a rarely executed statement, one can further analyze these statement and generate testbenches to activate them and analyze the responses.

The statement hardness analysis for b05 is shown in Figure 7 presents the number of lines with significant large statement hardness ranging from 1.13E+04 to 4.42E+05. To prevent Trojan insertion in a circuit at the behavioral level, it is possible to define a specific threshold on statement weight (statement hardness) and modify the circuit such that the statement weight (statement hardness) of all statements stands above (below) the threshold.

The detail analysis reveals the statement hardness depends on both the value range and the control flow. Defining a condition block on signals with a wide value range increases the statement hardness of statements inside the condition block. Nested condition blocks may increase statement hardness of interior statements as well. In above examples, although the statements with the highest statement hardness in b01 and b03 are both executed when four conditions are met, the execution of the statement in b03 is limited over the larger value range, which increases its statement hardness 3 times. On the other hand, for b04 benchmark, we observe that the statement with the highest statement hardness is executed when five conditions are met, while its statement hardness is the minimum among all the benchmarks due to the limited value range.

Based on the vulnerability analysis, the minimum observability for all benchmarks is 0; therefore, it is possible to implement behavioral Trojans with zero detectability. b01 has one internal signal, $stato$, to control a finite state machine with eight states, while $stato$ has zero observability. It is possible to implant a behavioral Trojan by extending the range of $stato$ and defining extra states (Trojan states). The Trojan can direct the finite state machine to Trojan states and mount a malicious attack. Similarly, b02 consists of one state machine, and the output is assigned to constant values. It is possible to extend the state machine and control the output signal under rare conditions. b03 grants access to resources through a finite state machine. A behavioral Trojan with zero detectability can be implemented by extending the finite state machine and putting a request in starvation when the Trojan gets activated. Some primary inputs of b04

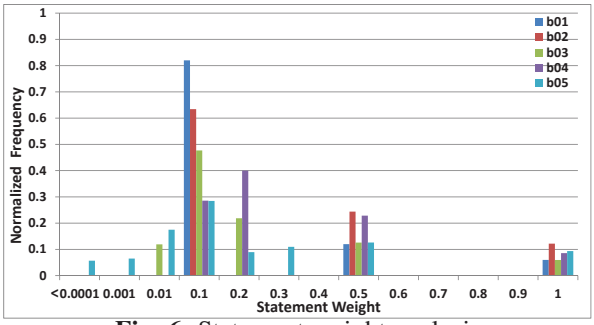

Fig. 6. Statement weight analysis.

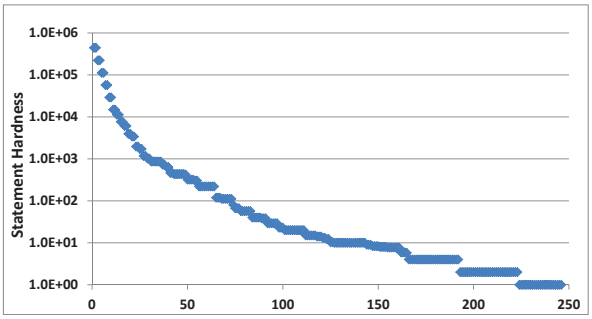

Fig. 7. Statement hardness for b05.

are not observable; hence, they can be modified internally under rare conditions to implant a Trojan attack. With the minimum and maximum observability of 0 and significant high statement hardness, it is possible to insert different Trojans in b05 in deep condition blocks with very low detectability to manipulate memory contents or propagate erroneous values to the outputs.

It should be noted that similar analysis can be easily applied to complex and hierarchical circuits. The execution time is a linear function of circuit size, thus the analysis will run extremely fast even on very large SoCs. The significance of our vulnerability analysis is that it determines which parts of a circuit are hard to activate without needing testbench. While developing a testbench to examine all control and data paths is a challenging task by itself, this vulnerability analysis is based on weighted value range. The weighted value range exactly determines the range of values under which a statement executes. This information can be used to develop a testbench intended to exercise potential Trojan locations.

At each level of the design process, particular measures are required to prevent Trojan insertion and to increase Trojan detection probabilities. To increase the probability of detecting behavioral Trojans, circuit statements with high statement hardness and signals with low observability should first be distinguished. Then, some techniques should be employed to ease executing statements with high statement hardness and to facilitate monitoring signals with low observability. For example, adding extra control statements accessible though primary inputs makes the execution of statements with high statement hardness more frequent. Similarly, passing low observable signals to primary outputs under a new set of statements and conditions could limit the capabilities of an adversary in implement a Trojan.

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a vulnerability analysis based on a weighted value range at the behavioral level to determine which parts of a circuit are more susceptible to Trojan insertion. To do so, the hardness of executing any statement (statement hardness), the observability of any internal signal, and the primary input were analyzed. The analysis revealed that the control flow and the value range induce statement hardness and observability. Detectability of behavioral Trojans was quantitatively determined, based on their statement weight and the observability of their target signals. To continue this work, we will apply our technique to more complex and hierarchical circuits. The work will be extended by introducing techniques to reduce the vulnerability of a behavioral circuit to Trojan insertion.

V. REFERENCES

[1] M. Tehranipoor and C. Wang, Introduction to Hardware Security and Trust, Springer, August 2011.

[2] M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," *IEEE Design and Test of Computers*, pp. 10-25, 2010.

[3] M. Li, A. Davoodi, and M. Tehranipoor, "A sensor-assisted self-authentication framework for hardware Trojan detection," in Proc. of the *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE12)*, pp. 1331-1336, 2012.

[4] C. Lamech, J. Aarestad, J. Plusquellic, R. Rad, and K. Agarwal "REBEL and TDC: Two embedded test structures for on-chip measurements of within-die path delay variations," in Proc. of the *IEEE/ACM International Conference on Computer-Aided Design (ICCAD11)*, pp. 170-177, 2011.

[5] X. Zhang and M. Tehranipoor, "RON: an on-chip ring oscillator network for hardware Trojan detection," in Proc. of the *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE11)*, pp. 1-6, 2011.

[6] L. Kim and J. Villasenor "A system-on-chip bus architecture for thwarting integrated circuit Trojan horses," *IEEE Transaction on Very Large Scale Integration (TVLSI) SYSTEMS*, Volume. 19, Issue. 10, pp. 1921-1926, 2011.

[7] M. Banga and M.S. Hsiao, "ODETTE: A non-scan design-for-test methodology for Trojan detection in ICs," in Proc. of the *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST11)*, pp. 18-23, 2011.

[8] H. Salmani and M. Tehranipoor, "Layout-aware switching activity localization to enhance hardware Trojan detection," *IEEE Transactions on Information Forensics and Security (TIFS)*, Volume: 7, Issue: 1, pp. 76-87, 2012.

[9] H. Salmani M. Tehranipoor, and J. Plusquellic, "A novel technique for improving hardware Trojan detection and reducing Trojan activation time," *IEEE Transactions on Very Large Scale Integration (TVLSI) Systems*, Volume: 20, Issue: 1, pp. 112-125, 2012.

[10] J. Aarestad, D. Acharyya, R. Rad, and J. Plusquellic, "Detecting Trojans through leakage current analysis using multiple supply pad I_{DDQ}s," *IEEE Transactions on Information Forensics and Security (TIFS)*, Volume: 5 , Issue: 4, pp. 893-904, 2010.

[11] J. Zhang, H. Yu, and Q. Xu, "HTOutlier: Hardware Trojan detection with side-channel signature outlier identification," in Proc. of the *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST12)*, pp. 55-58, 2012.

[12] S. Wei, S. Meguerdichian, and M. Potkonjak, "Malicious circuitry detection using thermal conditioning," *IEEE Transactions on Information Forensics and Security (TIFS)*, Volume: 6, Issue: 3, pp. 1136-1145, 2011.

[13] C. Lamech and J. Plusquellic, "Trojan detection based on delay variations measured using a high-precision, low-overhead embedded test structure," in Proc. of the *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST12)*, pp. 75-82, 2012.

[14] Y. Shiyanovskii, F. Wolff, A. Rajendran, C. Papachristou, D. Weyer, and W. Clay, "Process reliability based Trojans through NBTI and HCI effects," in Proc. of the *NASA/ESA Conference on Adaptive Hardware and Systems (AHS10)*, pp. 215-222, 2010.

[15] X. Wang, S. Narasimhan, A. Krishna, T. Mal-Sarkar, and S. Bhunia, "Sequential hardware Trojan: Side-channel aware design and placement," in Proc. of the *IEEE 29th International Conference on Computer Design (ICCD11)*, pp. 297-300, 2011.

[16] G. T. Becker, A. Lakshminarasimhan, L. Lang, S. Srivathsa, V.B. Suresh, and W. Burelson, "Implementing hardware Trojans: experiences from a hardware Trojan challenge," in Proc. of the *IEEE 29th International Conference on Computer Design (ICCD11)*, pp. 301-304, 2011.

[17] X. Zhang, N. Tuzzio, and M. Tehranipoor, "Red Team: design of intelligent hardware Trojans with known defense schemes," in Proc. of the *IEEE 29th International Conference on Computer Design (ICCD11)*, pp. 309-312, 2011.

[18] W. H. Harrison, "Compiler analysis of the value ranges for variables," *IEEE Transactions on Software Engineering,* Volume: SE-3, Issue: 3, pp. 243-250, 1977.

[19] ITC99 Benchmark, *http://www.cerc.utexas.edu/itc99-benchmarks/bench.html*