An Analysis of Differences between Trojans inserted at RTL and at Manufacturing with Implications for their Detectability

Samantha Pham[*], Jennifer L. Dworak[§], and Theodore W. Manikas[§]

[*]University of Santa Clara, Santa Clara, CA, USA

[§]Southern Methodist University, Dallas, Texas, USA

Abstract

As the design and manufacturing process has continued to fracture across multiple countries and companies, there is increasing concern that malicious hardware may be inserted into mission critical systems. Hardware Trojans are malicious changes to a circuit or system that compromise its functionality. They may create errors, leak information, or even destroy the chip. Much previous research has focused on Trojans inserted at the manufacturing stage through changes in the masks. However, malicious changes at the RTL are also possible and may be even more damaging as they may affect multiple generations of chips. Unfortunately, methods used to detect Trojans after manufacturing are likely to be entirely ineffective if Trojans are inserted earlier in the design cycle. Thus, research focused on Trojans inserted at the RTL or gate level is needed. In this paper, we will discuss several example Trojans that may be inserted in 3rd party IP and will analyze their stealth and effect on circuit complexity. Unlike most Trojans inserted at manufacturing, Trojans inserted at the RTL may decrease this complexity. We will then explore the ability of ATPG (Automatic Test Pattern Generation) test sets to aid in the detection of these Trojans.

1. Introduction

Over the past several years, the Department of Defense has expressed significant concern that circuits used in military applications are becoming increasingly susceptible to compromise through the insertion of hardware Trojans and the counterfeiting of parts. Hardware Trojans consist of malicious changes to a design that alter its functionality in an adverse way. For example, a Trojan may cause an error in the outputs for particular inputs or environmental conditions. Alternatively, it could leak secret information, such as the encryption key, for particular input sequences. In other cases, it could cause the chip to reset or even destroy itself. Because the impact of a hardware Trojan could be catastrophic in a mission-critical application, steps must be taken to either prevent the insertion of Trojans or to detect Trojans once they have been inserted.

Trojans may be theoretically be inserted at any stage of the design cycle from the creation of the specification through manufacturing [1], [2]. However, the majority of the research to date has focused primarily on Trojans inserted at manufacturing. The large percentage of parts manufactured overseas provides new opportunities for foreign agents or organizations to compromise chips by changing the masks used to make them. It is assumed that these Trojans would generally consist of small amounts of additional logic. Unlike defects, they would be intelligently inserted with the intention of making them difficult or impossible to detect with the standard logic tests applied in manufacturing. For example, Trojan activation might occur only with a very rare combination of internal signal values that are unlikely to be produced during testing. As a result, much previous research has focused on the impact the Trojan would have on power or delay even when the Trojan is not entirely activated. Multiple statistical techniques have been proposed to separate such changes from normal process variations (e.g. [3–5]).

However, instead of inserting Trojans at the manufacturing stage, Trojans could also be inserted at the RTL in third party IP or by a malicious insider. In fact, insertion at the RTL provides multiple advantages to the attacker. For example, the complex reverse engineering that would need to be done to insert meaningful Trojans at the mask level would likely be easier or unnecessary when the RTL is available to the attacker. In addition, every copy of the device—possibly over multiple generations of parts—would contain the Trojan, making the effort and risk of inserting it more worthwhile. Furthermore, because power and timing analysis tools would always operate on a version of the circuit that already contains the Trojan, they cannot be used for Trojan detection. However, ATPG (Automatic Test Pattern Generation) tests will become much more useful at the RTL.

Trojans inserted at the RTL also may differ from Trojans inserted at the mask level in another important way. Specifically, although Trojans inserted at the mask level often are assumed to add gates to a logic circuit, at the RTL, a particularly stealthy Trojan is more likely to remove functional complexity from the circuit. In some cases, they may also be capable of manipulating internal state machines so that the functional effect of the Trojan trigger does not appear for multiple cycles after triggering has occurred—making them even more difficult to detect.

In this paper, we will describe a sample of Trojans inserted at the RTL. We will explore how combinational and sequential Trojans may either add complexity to the circuit or remove complexity from the circuit. We will also explore how standard ATPG tools can be harnessed to create test sequences that will promote Trojan activation and detection under different circumstances.

2. Previous Work

Although most of the work in Trojan detection has focused on the manufacturing stage, there has also been recent work on hardware Trojan detection at the RTL. For example, Hicks et al. [6] identify two possible stages where Trojans can be inserted at the pre-fabrication level: either by a rogue employee, or in the software of third-party IP. The assumption is that the Trojan is inserted during the design phase at the RTL level and that it is digitally triggered. Hicks' approach uses design verification to identify "suspect" circuitry as possible hardware Trojans; a suspect circuit is one that is not used or activated by the design verification tests. The method was tested on a variation of the SPARC processor circuit, with Trojan models developed by the authors.

More recently, the authors of [7] also focused on Trojan detection in 3rd party IP through also focused on using design verification techniques—both formal and simulation-based, as well as limited sequential ATPG to identify suspect circuit locations and attempt to detect Trojans in an RS-232 circuit.

Banga and Hsaio [8] also used equivalence checking to identify possible hardware Trojans in gate level circuits. Their method compares a "suspect" circuit with a "spec" circuit (an unoptimized circuit quickly generated from the original system specification). The assumption is that the Trojan is inserted during the design phase at either the RTL or gate level, and that it is digitally triggered. They further assume that triggering is rare and use both functional patterns and an n-detect test set to remove faults from consideration that they consider too easy to detect. A SAT (Satisfiability) solver is then used to check for equivalence among faulted, unfaulted, and "spec" versions of the circuit. Finally a radius-based analysis is used to further refine the search space for Trojans.

The authors of [9] inserted Trojans into RTL code that was programmed into an FPGA. They considered the stealth of their design to be based on the number of hardware resources (LUT's and FF's)—indicating that the design was not made significantly bigger, as well as the power drawn by the FPGA.

However, to the best of our knowledge, no one has yet explicitly explored the differences in the types of Trojans that may be inserted at the RTL, how they affect circuit complexity, and how an attacker may make RTL/gate level Trojans at the design stage or in 3rd party IP particularly stealthy. In this paper, we will explore some of the characteristics such Trojans may have and what may be required to detect them.

3. Inserting Trojans at Manufacturing

Logical Trojans inserted at the manufacturing stage are generally assumed to consist of two components: a trigger and a payload. For example, consider Fig. 1. Assume that this is a part of a much larger circuit and that in this circuit B and C are highly skewed toward logic 1's. The Trojan corresponds to the shaded gates. The NOR gate forms the trigger. When both of the inputs to this gate are set equal to a logic 0, the output of the NOR gate becomes 1, and the Trojan is activated. In contrast, the XOR gate corresponds to the payload. It is spliced into a normal signal line of the circuit, and when the Trojan is activated, it complements the value on that signal line—causing an error.

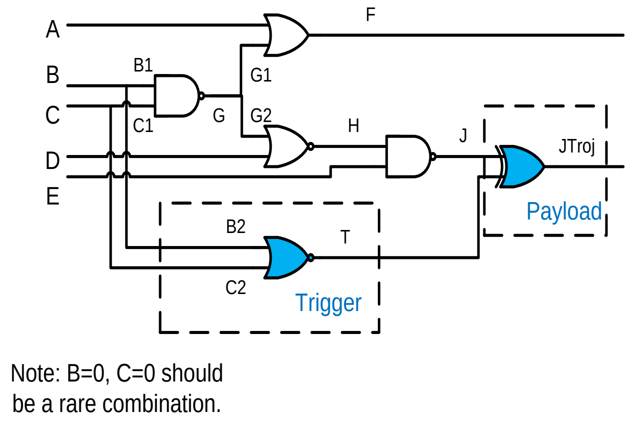

Figure 1

Although this is a very simple example, much more complex forms of Trojans are possible. For example, Trojans may be designed with counters so that they only trigger once a given pattern has been seen multiple times. Instead of inserting a simple one-bit error, they may totally change the output—for example, bypassing the encryption hardware and placing the plaintext on the data bus.

In general, Trojans inserted at the mask level share certain characteristics:

1) An attacker who wants his Trojan to have a functionally important impact must reverse engineer the design from the GDSII data. He needs to discover what signals to alter and which signals (if any) should be used to create the trigger.

2) The attacker must ensure that the overall size of the circuit does not increase. Any extra gates and routes should be placed in such a way as to leave the majority of the mask unchanged.

3) At the mask level, a Trojan may be made almost arbitrarily difficult to detect with traditional ATPG test patterns. Any extra logic added by the attacker will not appear in the netlist used to create the patterns. As a result the Trojan cannot be explicitly targeted during ATPG. An intelligent attacker can thus make it almost impossible to fortuitously detect the Trojan if the trigger or observation conditions are sufficiently difficult to satisfy. In addition, dynamic Trojans that make use of counters may be impossible to detect through structural scan-based ATPG as the attacker is highly unlikely to place the counter on the scan chain.

4) Although ATPG tests are unlikely to be able to detect the Trojan, other aspects of the Trojan, in particular power draw and delay, are likely to change whether or not the Trojan is explicitly active. For example, assume that a trigger corresponds to a 10-input AND gate. It may be very difficult to activate the AND gate by chance. However, if it is made from multiple smaller gates, the individual gates may switch even if the large AND gate does not. Added fanout from signal lines used to create the trigger and the insertion of additional gates into circuit paths to form the payload also add to circuit delay. As a result, multiple researchers have studied methods for the detection Trojans based upon the power or delay that they add to the circuit (e.g. [4–5,9-12]).

5) Finally, although this paper is currently focusing on Trojans that actively change the circuit logic and functionality, it is also possible to include non-logical Trojans at the manufacturing stage. For example, a Trojan could consist of an antenna inserted into an unused space of the design that

radiates the value on a particular signal.

4. Inserting Trojans at the RTL

There are obviously similarities between Trojans inserted at the RTL (or the synthesized gate level) and Trojans inserted at manufacturing. However, in other ways they can be very different.

Some characteristics of Trojans inserted at the RTL include:

1) A Trojan writer who is not the primary designer may still need to reverse engineer part of the design in order to create a Trojan with the desired functional impact. However, in this case, the reverse engineering can be performed at a higher level of abstraction—likely making it considerably easier than extracting information from GDSII data.

2) This reverse engineering may not be needed if the Trojan writer is also the designer (and seller) of the third party IP.

3) Trojans at the RTL are likely to still be designed to be difficult to excite/observe with random or functional pattern sequences. For example, they may still trigger on rare combinations of circuit conditions. This means that they will likely be very difficult to trigger with random/weighted random pattern sequences traditionally run for design verification.

4) One way other researchers have suggested could be used to identify potential Trojan circuitry in RTL code involves finding portions of the code with low code coverage during design verification. The assumption is that the Trojan code would be written to be so difficult to activate that it the lines of code corresponding to the Trojan would not be executed. However, a good Trojan writer could write a Trojan such that even if every statement in the code was executed at least once, the Trojan might still not be triggered. For example, consider the following pseudo RTL code:

```
Assign f2 = data_now_calculated & (A & B & C & D & E)’

Assign data_bus = calculated_data & (f2 XNOR data_now_calculated) +
              key & (f2 XOR data_now_collected)

Assign data_ready = data_now_calculated
```

   In this piece of code, the variables A, B, C, D, and E represent different events that together comprise the Trojan trigger condition. If all of the events are asserted (i.e. the variables are equal to 1), then the new variable f2 is set equal to 0. Otherwise, f2 is equal to “data_now_calculated.” To avoid branch structures, the data bus is assigned its value through a form of Shannon’s expansion theorem. If the value of f2 is the same as the value of “data_now_calculated,” as should almost always be true (and is always true when the Trojan is inactive), the value assigned to the data_bus is the expected “calculated_data.” However, if all the events are asserted, f2 will be different from the value of data_now_calculated (when that signal is asserted), and instead of the calculated data, the key will be placed on the data_bus and potentially available to a saboteur.

   Clearly, every line of this code will be executed by any simulation that enters this section. Thus, line coverage will be complete. There are no branches, so branch coverage will not be affected. Finally, every individual variable (aside from the key) will toggle on a regular basis in this or another part of the code—providing high toggle coverage. Thus, it is very possible that design verification sequences that attain high coverage according to traditional metrics will never activate this Trojan.

5) Trojans inserted at manufacturing generally add logic to the circuit. To some extent, this adds to the circuit complexity. In contrast, Trojans at the RTL may become stealthier if they can remove logic from the design and thus decrease complexity. In addition, Trojans at the RTL may become stealthier by delaying the Trojan effect for multiple clock cycles after the trigger condition has been met.

5. Making Trojans Difficult to Detect Through ATPG

Trojans consisting of extra gates inserted at manufacturing are hard to activate with ATPG patterns because the netlist used to generate the fault list does not contain the trigger logic. It is therefore impossible to for the ATPG tool to deterministically trigger the Trojan. For example, consider again the example circuit now shown in Figure 2. The stuck-at fault T sa0 will not be located in the fault list because none of the Trojan circuitry was in the original netlist. As a result, the ATPG tool is incapable of targeting this fault and activating the Trojan deterministically. In contrast, removing logic from a circuit is likely to be relatively easy to detect with traditional structural tests if the Trojan is inserted at the mask level and the original Trojan-free circuit is used to generate the tests. For example, if the Trojan writer chose to remove the OR gate with inputs A and G1 and directly connected A and F, a test for G1 sa0 would detect the error.

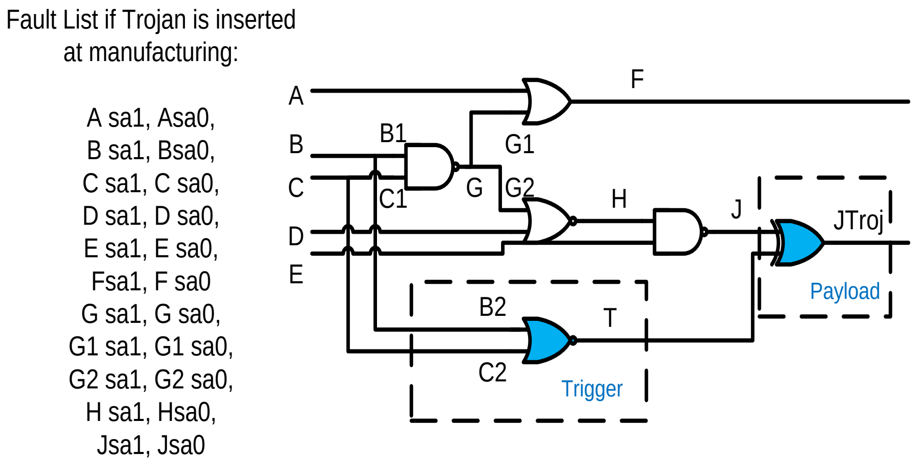

Figure 2: ATPG Netlist when Trojan is inserted at Manufacturing

The scenario is very different for Trojans inserted at the design stage. When Trojans are inserted in the RTL or gate level version of a design, they are inserted directly into the netlist that will be used to generate tests. As a result, the Trojan logic will appear in the fault list. If the ATPG tool generates a test set that attains 100% fault coverage, then a Trojan consisting of extra logic will be guaranteed to be activated and the effect propagated to an output or flip flop. For example, for the circuit in Figure 3, if the Trojan were inserted at the RTL instead of at the gate level, the input pattern XX00XX would be deterministically selected as one of the tests so that T sa0 could be detected. Detection of the Trojan then requires determining the functional implication of this test and the fault effect.

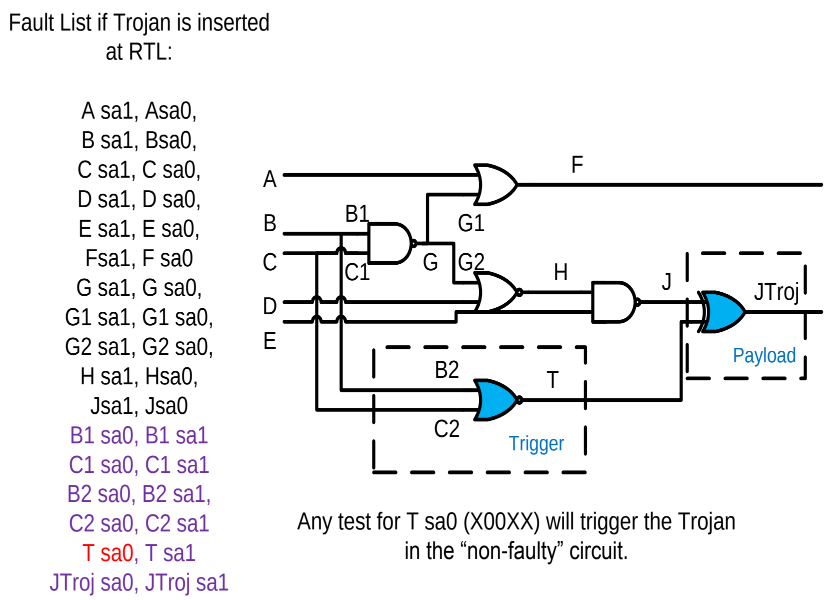

Figure 3: ATPG Netlist when Trojan is inserted at RTL.

However, there is another way a Trojan writer may choose to enter a Trojan at the RTL. Specifically, in some cases, the attacker may be able to insert a Trojan by removing logic. For example, For example, consider the following example circuit containing a Trojan:

```
module missileDet_T(detonate,inputData);
     input [31:0] inputData;
     output detonate;

     parameter detSeq = 32'h53756B75;
     parameter Trojan = 32'h53756B65;

     assign detonate = (inputData == detSeq || inputData ==
Trojan)?1:0;

endmodule
```

We created this Trojan benchmark, RWT08a, as a combinationally-triggered Trojan inspired by an example from [9]. In [9], the Trojan replaces all inputs corresponding to "Moscow" with "Boston." Thus, if a message is sent to "target Moscow," it is changed to "target Boston." In our experiments, we created a simple missile controller that detonates the missile when a 32-bit pattern is input to the circuit. The Trojan is triggered when an alternate pattern, which differs from the original pattern by only 1 bit, is input to the circuit. Thus, there are two possible patterns that can detonate the missile: the original (intended) pattern and the "secret" (Trojan) pattern. The Verilog code for this circuit is shown above, where the original pattern is "detSeq" and the Trojan pattern is "Trojan". Note that minimal code is required to implant the Trojan into the circuit and that even when the Trojan is not activated, every line in the circuit may be executed.

In our original implementation of this Trojan, the detSeq differed significantly from the Trojan sequence. This led to the addition of extra logic to serve as the Trojan trigger. However, in this version of the Trojan, the two sequences only differ by a single bit. This means that the 5th bit from the right becomes a don't care, and the detection logic for the sequence becomes simpler. There is simply one less input needed on the AND gate that identifies the detonation sequence. As a result, there is no trigger line present in the circuit, and there is no fault that ATPG can explicitly target to activate the Trojan.

More generally, good functional locations for an attacker to insert a Trojan in a logic function at the RTL would correspond to boundaries between 0's and 1's on a K-map. If the boundary is erased or moved by changing both sides to either 0 or both sides to 1, this is likely to allow the circuit to be further simplified. The error caused by the Trojan will not be able to be explicitly targeted by a testing tool, and the change may be difficult-to-detect with simulation-based verification methods, especially if this boundary is not associated with a known corner case. Of course, the changed minterm should still be unlikely to be hit by chance during functional test and verification. If not, it will likely be detected anyway unless other steps were taken to conceal its effects.

Finally, Trojans may become even more difficult to detect if the Trojan effect is not seen for multiple cycles after the Trojan was triggered. For example, consider again the missile detonation circuit. However, in this case assume that a particular string must be received a character at a time over several clock cycles to cause the missile to detonate. Specifically, the sequence "FIRE" must be received over 4 clock cycles. If any other sequence is received, the machine returns to the waiting state. A state machine corresponding to this behavior is shown in Figure 4.

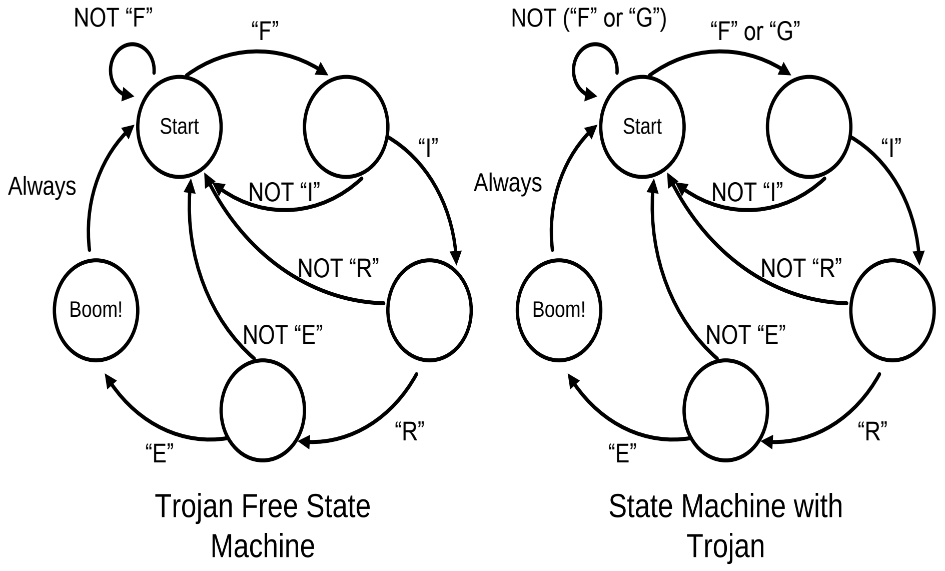

Figure 4: Trojan Inserted into State Machine

The Trojan-free machine is shown on the left. In the circuit containing the Trojan, an alternate sequence may cause the detonation as well: "GIRE." Because the letters "F" and "G" in ASCII only differ by one bit, the next state logic leaving the "Start" state is simpler than the original Trojan-free logic. Once again, there is no trigger signal for the ATPG tool to target. Furthermore, although a faulty transition may occur between the "Start" state and the 2nd state, this does not affect a true output (i.e. the detonator in this case), but only state elements. This makes the determination of the functional impact of even a test pattern that does activate the Trojan more difficult. We must either simulate for four clock cycles, or we must have detailed information about what constitutes a valid next state in each case. Of course, even more complicated Trojans are possible. For example, a single Trojan could involve multiple interacting state machines.

6. Experiments

In the following experiments, our goal was to explore the ability of the Synopsys ATPG tool TetraMAX [13] to generate test patterns that would both excite and propagate Trojan effects. The circuits were written in Verilog and then synthesized to obtain a gate level version. We first studied two combinational circuits with Trojans. RWT08 added two Trojan triggers to the circuit—each of which added logic. The second Trojan RWT08a (already described in previous sections), made the logic simpler. Table 1 provides information regarding the test sets generated and the ability of the patterns to activate the Trojan.

Table 1: Combinational Circuits with Trojans

Trojan	# of Gates	Function	#of Faults	# Test Pattern s	CPU Time	Fault Coverage	Sequences Detected?	
							Troj	Good
RWT08	26	Bad Output	274	55	0.00	100	Yes	Yes
RWT08a	11	Bad Output	170	37	0.00	100	Yes	No

In the case of the Trojan that added logic (RWT08), both the Original (Good) and Trojan (Bad) inputs combinations were generated and applied during test. Thus, the Trojan was detected. In the case of the second circuit, the logic became simpler when the Trojan was added. One of the inputs became a “don’t care.” This means that the ATPG tool didn’t have any preference for selecting the Trojan or Trojan-free input combination when generating a test for the detonation output stuck-at 0. That bit was randomly filled. In this case, we were lucky. The Trojan input combination was randomly selected, and we detected the Trojan. The input combination originally intended by the designer was not used during test.

Next, we began investigating sequential circuits. FSM_Jenn was a state machine similar to that shown in Figure 4. Once again an alternate Trojan sequence was inserted. This sequence differed from the original by only 1 bit in the first letter of the sequence. Table 2 shows the results obtained when sequential ATPG patterns were generated with TetraMax. In this case, both the Trojan sequence and the original sequence were used. This is fortunate as there is no guarantee that this would necessarily have been the case.

Table 2: Trojan Detection with Sequential ATPG

Trojan	# Gates	# of Flip Flops	Function	# of Faults	# Test Patterns	CPU Time	Fault Coverage	Detected?	
								Good	Troj
FSM_Jenn	18	3	Bad Output	172	24	50.84	97.67	Yes	Yes

Unfortunately, sequential ATPG is time consuming (even for relatively small circuits.) Thus, up to a point, it is preferable to insert scan chains into the design and use combinational scan-based ATPG instead. Results from some of these experiments are shown in Table 3. Once again, FSM_Jenn could have asserted the output for either the original intended sequence or the one corresponding to the Trojan. Once again we were lucky, and the Trojan was selected.

We then repeated this experiment for two RS232 Trojan circuit benchmarks that are freely available from TrustHub [14]. The first Trojan (RS232-TG09C14PI0) contained 8 combinational gates. The trigger occurred when 20 signals were high and the payload changed two bits of the output.

The second Trojan (RS232-TG1AS1BPI0) contained 4 flip-flops and 10 combinational gates. The trigger occurred when 24 signals were high, and the payload changes one bit of the output. Note that the TrustHub benchmarks were generally designed for detecting Trojans inserted at manufacturing. Thus, the flip-flops were not placed on the scan chain in the original benchmark. Because our Trojans are inserted in the RTL, the scan insertion tool would see the flip-flops and would place them on the scan chain. Thus, they were on the scan chain in our experiment.

As one can see from Table 3, both RS232 Trojans were easily detected.

Table 3: Sequential Trojans Circuits Tested with Scan-Based ATPG

Trojan	# Gates	# of Flip Flops	Function	# Faults	# Test Patterns	CPU Time	Fault Coverage	Detected?	
								Good	Troj
FSM_Jenn	18	3	Bad Output	172	31	0.00	100	No	Yes
RS232-TG09C14PI0	152	58	Bad Output	71	71	0.01	98.40	Yes	
RS232-TG1AS1BPI0	65	62	Bad Output	1886	69	0.01	98.56	Yes	

7. Conclusions

Trojan detection and insertion have important differences as one moves from considering Trojans inserted at manufacturing to those inserted in the RTL. In particular, Trojans inserted in the RTL become stealthier when they remove logic from the circuit. In the future, we will expand this investigation to new larger circuits and a variety of Trojans. We will also further explore the tradeoffs between different types of sequential and combinational ATPG for effective Trojan detection.

8. References

[1] S. Adee, "The Hunt For The Kill Switch," Spectrum, IEEE DOI - 10.1109/MSPEC.2008.4505310, vol. 45, no. 5, pp. 34–39, 2008.

[2] M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," Design & Test of Computers, IEEE DOI - 10.1109/MDT.2010.7, vol. 27, no. 1, pp. 10–25, 2010.

[3] C. Lamech, R. M. Rad, M. Tehranipoor, and J. Plusquellic, "An Experimental Analysis of Power and Delay Signal-to-Noise Requirements for Detecting Trojans and Methods for Achieving the Required Detection Sensitivities," Information Forensics and Security, IEEE Transactions on DOI - 10.1109/TIFS.2011.2136339, vol. 6, no. 3, pp. 1170–1179, 2011.

[4] Yier Jin and Y. Makris, "Hardware Trojan detection using path delay fingerprint," in Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on, 2008, pp. 51–57.

[5] R. M. Rad, Xiaoxiao Wang, M. Tehranipoor, and J. Plusquellic, "Power supply signal calibration techniques for improving detection resolution to hardware Trojans," in Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on, 2008, pp. 632–639.

[6] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith, "Overcoming an Untrusted Computing Base: Detecting and Removing Malicious Hardware Automatically," in Security and Privacy (SP), 2010 IEEE Symposium on, 2010, pp. 159-172.

[7] Xuehui Zhang and M. Tehranipoor, "Case study: Detecting hardware Trojans in third-party digital IP cores," in Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on, 2011, pp. 67-70.

[8] M. Banga and M. S. Hsiao, "Trusted RTL: Trojan detection methodology in pre-silicon designs," in Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on, 2010, pp. 56-59.

[9] Y. Jin, N. Kupp, and Y. Makris, "Experiences in Hardware Trojan design and implementation," in Hardware-Oriented Security and Trust, 2009. HOST '09. IEEE International Workshop on, 2009, pp. 50–57.

[10] Y. Alkabani and F. Koushanfar, "Consistency-based characterization for IC Trojan detection," in Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on, 2009, pp. 123-127.

[11] S. Wei, S. Meguerdichian, and M. Potkonjak, "Gate-level characterization: Foundations and hardware security applications," in Design Automation Conference (DAC), 2010 47th ACM/IEEE, 2010, pp. 222-227.

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

[13] TetraMAX from Synopsys: http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/TetraMAXATPG.aspx

[14] TrustHub: http://trust-hub.org/