High-Level Synthesis for Run-Time Hardware Trojan Detection and Recovery

Xiaotong Cui
College of computer Science
Chongqing University
Chongqing, China 400044
xiaotong.sd@gmail.com

Kun Ma
Department of ECE
University of Illinois
Chicago, IL, 60607
makun718@gmail.com

Liang Shi
College of computer Science
Chongqing University
Chongqing, China 400044
shi.liang.hk@gmail.com

Kaijie Wu
College of computer Science
Chongqing University
Chongqing, China 400044
kaijie@gmail.com

ABSTRACT
Current Integrated Circuit (IC) development process raises security concerns about hardware Trojan which are maliciously inserted to alter functional behavior or leak sensitive information. Most of the hardware Trojan detection techniques rely on a golden (trusted) IC against which to compare a suspected one. Hence they cannot be applied to designs using third party Intellectual Property (IP) cores where golden IP is unavailable. Moreover, due to the stealthy nature of hardware Trojan, there is no technique that can guarantee Trojan-free after manufacturing test. As a result, Trojan detection and recovery at run time acting as the last line of defense is necessary especially for mission-critical applications. In this paper, we propose design rules to assist run-time Trojan detection and fast recovery by exploring diversity of untrusted third party IP cores. With these design rules, we show the optimization approach to minimize the cost of implementation in terms of the number of different IP cores used by the implementation.

Categories and Subject Descriptors
B.7.m [**Hardware**]: Integrated Circuits–*Miscellaneous.*

General Terms
Experimentation, Security

Keywords
Hardware Trojan, detection and recovery, design for security, run time, IP, high-level synthesis

1. INTRODUCTION
The globalization of IC design and manufacture process raises serious concerns about security and trustworthiness. During the design and manufacture process, an adversary may insert hardware Trojan into ICs to trigger logic errors, leak sensitive information, degrade performance, etc. Ensuring that an IC is Trojan-free before it is deployed is challenging. Hardware Trojan is typically activated under extremely rare condition to keep stealthy. Activating Trojan during test time is a very difficult task because it is infeasible to generate an exhaustive set of test vectors to explore the huge Trojan space for detection. Instead, statistical approaches have been proposed for logic testing with some certain Trojan detection coverage [1], [2]. Besides logic testing, side-channel analysis has been proposed to detect Trojan as well. As the insertion of Trojan affects side-channel parameters of the host circuit such as power consumption and path delay, deviation of these parameters from expected values indicates the presence of Trojan [3], [4]. However, large process variation and measurement noise can mask the variation introduced by Trojan circuit, especially for small Trojans. Despite various test-time detection approaches, no approach guarantees detection of all possible hardware Trojan at test time.

The difficulty of test-time detection is further increased for IC designs with third party IP cores. Most of the existing test-time detection methods require a "golden" (i.e. Trojan-free) IC or functional model against which to compare the behavior of the IC-under-test. However, an IP-core vendor will only provide one IP core for a required function. There will not be a "golden" version against which the trustworthiness can be verified.

If a hardware Trojan in an IC escapes the test-time detection and the IC is deployed, it may cause unacceptable lost once activated at run time. This is intolerable to many mission-critical applications such as avionics, communications, finance, military, and etc. Run-time Trojan detection must be deployed to continuously monitor the host IC's behavior when it is running. Rajendran et al. [5] proposed a design approach that assists run-time Trojan detection. However their technique does not address how the circuit recovers from the abnormal state caused by an activated Trojan. It is important to note that for mission-critical applications, the infected ICs are expected to continue working correctly until they can be replaced. Hence recovery from abnormal state caused by activated Trojans is as important as detection. Trojan-induced errors have a different mechanism from environment-induced soft and hard errors, hence demand different detection and recovery methods than the traditional techniques.

In this paper, we present a new design approach that supports run-time Trojan detection and fast recovery. It exploits the diversity of IP cores and uses high-level scheduling and binding techniques to detect and recover hardware Trojans. To the best of our knowledge, this will be the first work that considers both run-time Trojan detection and recovery. We first propose a set of design rules that combine Trojan detection and fast recovery. We then optimize the number of IP cores from diverse vendors needed by a design by formulating the problem using Integer Linear Programming (ILP) method. The paper is organized as follows. Section 2 introduces the most recent work. Section 3 presents the design rules that support run-time Trojan detection and recovery. Section 4 formally defines the optimization problem and presents the ILP formulation. The experimental results and conclusions are given in Section 5 and Section 6, respectively.

DAC '14, June 01 - 05 2014, San Francisco, CA, USA

2. Related Work

Rajendran et al. proposed a design approach to support Trojan detection at run time [5]. Their approach takes advantage of IP cores from diverse vendors. Although no "golden" IP core is available for comparison to ensure trust, the outputs of two IP cores of the same functionality but from different vendors can be compared to discover abnormal behavior. IP cores from different vendors tend to have different implementations. It is unlikely that they both are Trojan-infected. Even if they are, the chance that their Trojans are activated by the same way is assumed to be extremely rare.

Two function-equivalent computations using IP cores from different vendors are carried to detect Trojan-induced errors. The two computations are referred to as normal computation (NC) and recomputation (RC) respectively. The NC is for the original function to be implemented and the RC is for Trojan detection. The outputs of NC and RC are compared with a mismatch indicating the detection of Trojan. A simple example is shown in Figure 1. Symbols +, × indicates general operations performed by IP cores. Different sources (i.e. vendors) of IP cores are shown in different colors. Any operation in NC and its corresponding operation in RC are performed by two IP cores from different vendors. If a Trojan in one IP core is activated, it will alter its output and hence the output of the computation where it resides (either NC or RC), and will be detected by comparing the outputs of NC and RC. In addition to detection, Rajendran et al. also suggest to reduce the chance of activating a Trojan by preventing collusion. Collusion could happen when two units from same vendor directly interact with each other – one unit could embed the trigger signal and pass it to the other unit at the direct downstream. With the direct interaction, the Trojan in the downstream unit could be triggered much more easily. Hence they suggest that any two operations where the output of one operation is the direct input of the other operation should be bound to IP cores from different vendors. And for the same reason, two operations which produce results as inputs to the same operation should be bound to IP cores from different vendors as well.

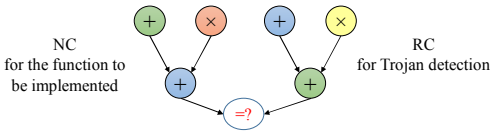


Figure 1. Trojan detection using IP cores from diverse vendors [5]

The design rules proposed by Rajendran et al. to support run-time Trojan detection are summarized as follows.

Rule 1 for detection: IP cores from diverse vendors are used to construct two computations (NC and RC) to detect the logic errors caused by Trojan. The same operation in two computations should be bound to IP cores from different vendors.

Rule 2 for detection: Operations which have the relation of parent-child or provide inputs to the same operation should be bound to IP cores from different vendors.

3. Trojan Detection and Recovery

While Rajendran's work helps run-time detection of Trojan, it does not address how the circuit recovers from the abnormal state caused by an activated Trojan. It is important to note that for mission-critical applications, the infected ICs are expected to continue working correctly until they can be replaced. Hence recovery from abnormal state caused by activated Trojans is as important as detection. Thereafter we will consider two phases of a circuit: In the Detection Phase, computations are redundant where NC is for the original function-to-implement and RC is to detect Trojan as shown above. Upon a Trojan detection, the circuit switches to the Recovery Phase where the computation is non-redundant. A novel high-level scheduling and binding technique is proposed to deactivate the Trojan and recover the circuit to normal state. In this section, we will first introduce the classification of the various types of hardware Trojans based on their trigger mechanisms and payload functions. We then clearly define the focus of this work, based on which we discuss the fault model of the Trojan-induced errors, followed by the proposed fast recovery approach.

3.1 Hardware Trojans

Hardware Trojans can be classified according to its trigger mechanism and payload function. The trigger logic monitors a set of inputs and activates the payload under a certain condition. Payload, once activated, will lead to malicious deviations from expected behavior. Trigger logic and payload logic can be digital or analog. In this paper, we consider digital Trojans which can affect the logic values.

There are two types of trigger mechanisms: combinational trigger logic and sequential trigger logic. Figure 2 (a) shows an example of hardware Trojan with combinational trigger logic. The payload is activated under the condition that A = 0, B = 0. Sequential triggered logic, on the other hand, activates the payload by a sequence of continuous operations. Figure 2 (b) shows a sequentially triggered Trojan based on a counter. The payload is activated when the counter reaches 2^k-1.

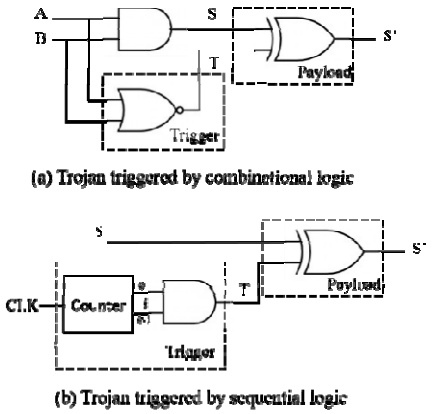
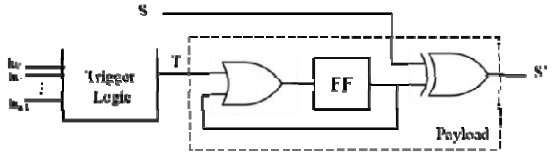


Figure 2. Trigger mechanisms of Hardware Trojan

The payload logic can be classified based on whether they can remember and keep their activation status. The payload logic shown in Figure 2 modifies internal signal using XOR gate. The activation of payload only depends on the logic value of trigger signal produced by trigger logic. In contrast, an example of payload logic with memory element is shown in Figure 3. Once activated, the payload logic can remember its activation status and keep it. Apparently, the payloads with memory element usually pose stronger effects on their host circuits by taking more silicon area, consuming more power, introducing more path delay, and producing persistent alterations to outputs. All of these significantly improve the chance of detecting them at the test time. Since this work is on run-time detection and recovery, we will focus on the Trojans that have payloads WITHOUT memory element that has a



better chance to escape from test-time detection.

Figure 3. Payload with memory element

The trigger logic, no matter on which mechanism it is based, takes a set of internal nodes and/or primary inputs as input. Therefore, inputs to trigger logic determine the activation status of Trojan. Only particular input values or sequence of input values will activate the Trojan. By manipulating the values of inputs to trigger logic, we can reset trigger signal T and then deactivate Trojan whose payload has no memory element. Moreover, activated Trojans may affect their host units in many ways depending on their designs [6], [7]. Among all these, the most straightforward and effective way is to alter the output of its host unit, which will be the focus of this work.

In summary, this work will try to detect and recover the Trojan that has the following properties:

- Its trigger mechanism could be combinational or sequential. Its trigger signal(s) will be set when the trigger conditions are met, and will be reset when the otherwise;
- Its payload has no memory element, and will be deactivated if its triggering signal is reset;
- Its payload function is to alter the main output of the host unit, i.e., causing logic errors;

3.2 Trojan-caused Fault Model

Although activated Trojan causes logic error, it cannot be modeled by traditional fault models as its mechanism is different from the environment-induced soft or hard errors. An environment-induced soft error, such as Single Event Upset, disappears after a period of time, and the affected unit is still healthy [9]. Hence a simple re-execution of the same computation using the same unit will recover the error. An environment-induced hard error, such as Single Event Latch-up, does not disappear, and the affected unit is deemed faulty and becomes unusable thereafter [10]. Recovering from hard errors requires more resource. A logic error caused by Trojan, on the other hand, will be persistent if its trigger condition remains true, and will disappear (deactivate) if its trigger condition becomes invalid. Here we define a new fault model to describe Trojan-caused logic errors: A Trojan-caused logic error will continuously introduce offsets to the output of its host unit until its trigger condition of the Trojan becomes invalid. Therefore, the logic error caused by an activated Trojan cannot be recovered by a simple re-execution that re-executes the same computations using the same units – most probably the triggering condition of the activated Trojan will remain valid in recovery. Hence we propose a novel recovery technique to invalidate the triggering condition by re-binding the operations carried by each unit in recovery, as shown in next subsection.

3.3 Design for Fast Recovery

Once a Trojan is detected at run time, we want to recover the normal operations as soon as possible. As discussed in section 3.2, simple re-execution using the same resources is ineffective to logic errors caused by Trojan. While identifying infected IP core and replacing it with a function-equivalent core from another vendor can recover the operations, but the identifying process may cost too much in terms of time and resources and hence prevent fast recovery [11], [12]. In order to recover normal operations fast and minimize the negative effects of Trojan, we temporarily tolerate the infected IP core by deactivating Trojan and use it in recovery.

It is reasonable to assume that, if an IP core contains a Trojan, all instantiations of the IP core will contain the same Trojan. Based on the design rules for run-time Trojan detection, an IP core will not be used to perform the same operations in two computations (i.e., NC and RC). Since the triggering condition of a Trojan is supposed to be a rare input or input sequence (otherwise they are hard to escape from the test-time detection), it is assumed that if an IP core contains a Trojan and the Trojan is activated by a certain input or input sequence in one operation, Trojans in the instantiations of the same IP core will not be activated as long as they are assigned to perform different operations in the DFG for NC or RC. Therefore, recovery can be achieved by ensuring that operations must be re-bound to the IP cores of the vendors that are different from those used in detection phase. New IP cores will be introduced if necessary. Figure 4 shows an example of recovery where × operations are re-bound by reusing IP cores while + operations are performed on new IP cores. Hence we have the following rule for recovery:

Rule 1 for fast recovery: During recovery, operations should be re-bound to the IP cores of the vendors that are different from those in detection phase.

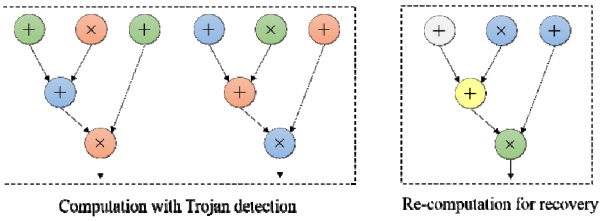


Figure 4. Fast recovery by re-binding operations to different IP cores

For any two operations in a Data Flow Graph (DFG), the inputs are said to be closely related if the difference between them is always small. The operation pairs with closely-related inputs could exist due to the properties of some algorithms such as DSP. In order to deactivate Trojan in recovery, the operation pairs with closely-related inputs can be treated as if they were the same operations. Hence the same rules will be applied.

Rule 2 for fast recovery: During recovery, an operation should be re-bound to the IP cores of the vendors that are different from its closely-related operations in detection phase.

The operation pairs with closely-related inputs can be identified by analyzing the algorithm or profiling input relations through a large set of test vectors generated by its targeting applications.

4. The Optimization Problem and the ILP Formulation

Design with untrusted IP cores should follow the two rules for detection from [5], and the two rules for recovery proposed here. While these design rules impose requirement on the diversity of IP cores and ensure the run-time Trojan detection and recovery, they also increase the purchasing cost – the license fees paid to IP vendors. The following example motivates this optimization problem.

The 5-operation DFG shown in the up-left corner of the Figure 5 is the function to be implemented, which is the NC. An RC needs to be implemented for Trojan detection, as shown in the up-right corner of the Figure 5. The NC and RC together are referred to as Detection Phase. A mismatch between the outputs of NC and RC indicates the detection of Trojan and will trigger the Recovery Phase, in which the original 5-operation DFG will be re-implemented.

In addition to the DFG to be implemented, the designer will be provided the maximal latency constraint for detection phase and

recovery phase, as well as the maximal silicon area constraint. Besides, the designer will also be provided with a list of vendors and the prices and area for each IP core they offer, as shown in Table 1. It is assumed that using multiple copies of a same IP core does not incur additional fee. The goal of the designer is to find, if exists, the optimal design that follows the rules for detection and the rules for recovery, meets the latency and area constraints, and incurs the minimal purchasing cost.

Table 1 Area and cost for each type of computational IPs

VENDOR	TYPES	AREA (unit cell)	COST (IP core license)
Ven 1	adder	532	$450
	multiplier	6843	$950
Ven 2	adder	640	$630
	multiplier	5731	$880
Ven 3	adder	763	$540
	multiplier	6325	$760
Ven 4	adder	618	$580
	multiplier	5937	$1000

The schedule with minimum cost is shown in Figure 5. The latency constraints for detection and recovery phases are 4 cycles and 3 cycles respectively, and the total area constraint is 22000, the optimal design is shown in Figure 5, which incurs total 4160 dollars purchasing cost.

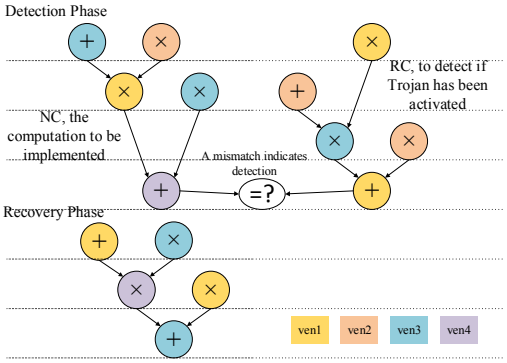


Figure 5. Motivational Example

Formally speaking, given a DFG-to-be-implemented, the latency constraints for both detection and recovery phases, the total area constraints, and a list of vendors and their IP cores, one wants to find the optimal design, if exists, that incurs the minimal purchasing cost while satisfies all the constraints.

4.1 ILP Formulations

The following notations and definitions in Table 2 will be used in the ILP formulations.

Table 2 Notations and Definitions

Notation	Definition
n	Total number of operations in the function-to-be-implemented, i.e., NC
ven	ven is the set of vendors from which IP cores can be purchased.
$\|ven\|$	Known variable, which is equal to the number of vendors.
τ	The set of different types of IP cores
$\|\tau\|$	Known variable, which is equal to the number of different types of IP cores of the same type. For instance, if we have one set of adders and one set of multipliers, then $\|\tau\|$=2
$\tau(t)$	The set of resources in t-th type of IP cores
$\|\tau(t)\|$	Known variable, which is equal to the number of resources in the *t*-th type of IP cores
$c(ven,\tau)$	A two-dimensional array, where $c(k,t)$ is a known variable representing the cost for purchasing IP core license of the *t*-th type of IP cores from *k*-th vendor. $1 \le k \le \|ven\|$, $1 \le t \le \|\tau\|$.
$\pi(ven,\tau)$	A two-dimensional array, where $\pi(k,t)$ is a known variable representing the area of a IP cores of *t*-th type from *k*-th vendor.
$\delta(ven,\tau)$	A two-dimensional array used to record the usage of some type of IP cores from one vendor, where $\delta(k,t)$ is 0-1 unknown variable, it is equal to 1 if and only if *t*-th type of IP cores from *k*-th vendor are used in computations. Otherwise, it is equal to 0. $1 \le k \le \|ven\|, 1 \le t \le \|\tau\|$.
$\varepsilon(ven,\tau,\tau(t))$	A three-dimensional array used to record which IP cores are used in our computations. $\varepsilon(k,t,m)$ is a unknown 0-1 variable. It equals to 1 if and only if the *m*-th IP cores of *t*-th type from *k*-th vendor are used. Otherwise, it equals 0. $1 \le k \le \|ven\|, 1 \le t \le \|\tau\|, 1 \le m \le \|\tau(t)\|$.
λ	Known variable, it is an upper bound of scheduling length. The scheduling length covers a schedule of detection phase and a schedule of recovery phase.
A	Known variable, it is an upper bound of the sum area of all IP cores used in detection and recovery schedule.
o_i	o_i represents operation i, $1 \le i \le n$
$ot(o_i)$	Known variable, it denotes the operation type of o_i. We may use 0 to represent addition, so if $ot(o_i)$ is an addition, then $ot(o_i)=0$ and o_i must be performed on an adder.
$e(o_i,o_j)$	0-1 known variable. It denotes the dependency between o_i and o_j. $e(o_i,o_j)=1$ represents that the execution of o_j depends on the output of o_i. Otherwise, $e(o_i,o_j)=0$
$D_{i,l,k,m}$ $D'_{i,l,k,m}$ $R_{i,l,k,m}$	$D_{i,l,k,m}$, $D'_{i,l,k,m}$, $R_{i,l,k,m}$ are unknown 0-1 variables which represent the schedule of original and redundant copies of o_i in detection phase and the schedule of original copy in recovery phase respectively. $D_{i,l,k,m}=1$ if and only if o_i is scheduled at step l and performed on *m*-th IP cores from vendor k, otherwise, $D_{i,l,k,m}=0$. With $D'_{i,l,k,m}$ and $R_{i,l,k,m}$, it is the same. ($1 \le l \le \lambda$, $1 \le k \le \|ven\|, 1 \le m \le \|\tau(t)\|$)
Z	Z is a very large positive integer.

In the ILP formulation, we will use the following theorem to translate an "if and only if" statement to linear form.

Theorem1.Let x, y be integer binary variables, the statement "x=1, then y =0" can be translated to the following linear form.

$$x - 1 \le y \le 1 - x \qquad (1)$$

Proof: If $x=1$, then $0 \le y \le 0$, so $y=0$ is true. On the other hand, if $x=0$, then $-1 \le y \le 1$, since y is a binary variable,

so $0 \le y \le 1$.

Theorem2.Let x, a, b, c, d be 0-1 binary variables. The statement "$x = 1$ if and only if $a + b + c + d \ge 1$" can be translated to the following linear form.

$$(a + b + c + d) / Z \le x \le a + b + c + d \qquad (2)$$

Proof: If $a + b + c + d \ge 1$, since Z is a very large positive integer, so $0 < x \le 4$, since x is a 0-1 variable, so $x = 1$ is true; On the other hand, if $x = 1$, $a + b + c + d \ge 1$ is true.

1) Operation scheduling constraints: There are three copies of each operation: two in Detection Phase with one in NC, one in RC, and one in Recovery Phase. These constraints ensure that each of three copies of an operation is scheduled once and only once. For all $1 \le i \le n$

$$\sum_{l=1}^{\lambda} \sum_{k=1}^{|ven|} \sum_{m=1}^{|\tau(t)|} \mathrm{H}_{i,l,k,m} = 1 \qquad (3)$$

H can be replaced by D, D' and R respectively. Therefore, (3) represents 3 constraints. It is the same with the following constraints with character H.

2) Dependency constraints: if there is an edge from o_i to o_j, it means the execution time of o_j must be greater the execution time of o_i. Thus, for all $e(o_i, o_j) = 1$

$$\sum_{l=1}^{\lambda} \sum_{k=1}^{|ven|} \sum_{m=1}^{|\tau(t)|} \left(\mathrm{H}_{i,l,k,m} \times l\right) + 1 \le \sum_{l=1}^{\lambda} \sum_{k=1}^{|ven|} \sum_{m=1}^{|\tau(t)|} \left(\mathrm{H}_{j,l,k,m} \times l\right) \qquad (4)$$

3) Detection Phase constraints: According to Rule 1 for detection, the original copy and its corresponding redundant copy cannot be performed on IP cores from the same vendors in detection phase. Thus, for all $1 \le i \le n$, $1 \le k \le |ven|$

$$\sum_{l=1}^{\lambda} \sum_{m=1}^{|\tau(t)|} (D_{i,l,k,m} + D'_{i,l,k,m}) \le 1 \qquad (5)$$

According to Rule 2 for detection, we need to consider three constraints. First, one parent node and its child cannot be performed on IP cores from the same vendor. For all $e(o_i, o_j) = 1$, $1 \le k \le |ven|$

$$\sum_{l=1}^{\lambda} \sum_{m=1}^{|\tau(t)|} (\mathrm{H}_{i,l,k,m} + \mathrm{H}_{j,l,k,m}) \le 1 \qquad (6)$$

Second, parents with the same child cannot be performed on IP cores from the same vendor. For all $e(o_i, o_b) = 1$, $e(o_j, o_b) = 1$ ($i \ne j$), $1 \le k \le |ven|$

$$\sum_{l=1}^{\lambda} \sum_{m=1}^{|\tau(t)|} (D_{i,l,k,m} + D_{j,l,k,m}) \le 1 \qquad (7)$$

4) Recovery Phase constraints: According to Rule 1 for fast recovery, IP cores carrying one copy of o_i in recovery phase and IP cores carrying copies of o_i in detection phase cannot come from the same vendor. Thus, for all $1 \le i \le n$, $1 \le k \le |ven|$

$$\sum_{l=1}^{\lambda} \sum_{m=1}^{|\tau(t)|} (D_{i,l,k,m} + D'_{i,l,k,m} + R_{j,l,k,m}) \le 1 \qquad (8)$$

According Rule 2 for fast recovery, if two operations o_i and o_j have close-related inputs, we have the following constraint: for $ot(i) = ot(j)$, if $D_{i,l,k,m} = 1$, then $D_{j,l,k,m} = 0$, and the constraint can be translated to the following linear form according to theorem 1

$$D_{i,l,k,m} - 1 \le R_{i,l,k,m} \le 1 - D_{i,l,k,m} \qquad (9)$$

$$D'_{i,l,k,m} - 1 \le R_{i,l,k,m} \le 1 - D'_{i,l,k,m} \qquad (10)$$

5) Usage constraint: The usage of each IP core of each type from each vendor can be recorded by the following linear formulation according to theorem 2, for all $ot(i) = t$, $1 \le k \le |ven|$, $1 \le m \le |\tau(t)|$,

$$\begin{aligned} &\sum_{i=1}^{n} \sum_{l=1}^{\lambda} \left(D_{i,l,k,m} + D'_{i,l,k,m} + R_{i,l,k,m}\right) / Z \le \varepsilon(k,t,m) \\ &\le \sum_{i=1}^{n} \sum_{l=1}^{\lambda} \left(D_{i,l,k} + D'_{i,l,k} + R_{i,l,k}\right) \end{aligned} \qquad (11)$$

Similarly, the usage of each type of IP cores from each vendor can be recorded by the following formulation, for all $ot(i) = t$, $1 \le k \le |ven|$,

$$\begin{aligned} &\sum_{i=1}^{n} \sum_{l=1}^{\lambda} \sum_{m=1}^{|\tau(t)|} \left(D_{i,l,k,m} + D'_{i,l,k,m} + R_{i,l,k,m}\right) / Z \le \delta(k,t) \\ &\le \sum_{i=1}^{n} \sum_{l=1}^{\lambda} \sum_{m=1}^{|\tau(t)|} \left(D_{i,l,k,m} + D'_{i,l,k,m} + R_{i,l,k,m}\right) \end{aligned} \qquad (12)$$

6) Area constraint: the sum of area of all IP cores used in computations cannot exceed the given area constraint A,

$$\sum_{k=1}^{|ven|} \sum_{t=1}^{|\tau|} \sum_{m=1}^{|\tau(t)|} \left(\varepsilon(k,t,m) \times \pi(k,t)\right) \le A \qquad (13)$$

7) Phase order constraints: the Recovery Phase cannot start until the Detection Phase finishes. Let o_n be the last schedule operation in detection phase, for all $1 \le i \le n$

$$\sum_{l=1}^{\lambda} \sum_{k=1}^{|ven|} \sum_{m=1}^{|\tau(t)|} D_{n,l,k,m} \le \sum_{l=1}^{\lambda} \sum_{k=1}^{|ven|} \sum_{m=1}^{|\tau(t)|} R_{i,l,k,m} \qquad (14)$$

$$\sum_{l=1}^{\lambda} \sum_{k=1}^{|ven|} \sum_{m=1}^{|\tau(t)|} D'_{n,l,k,m} \le \sum_{l=1}^{\lambda} \sum_{k=1}^{|ven|} \sum_{m=1}^{|\tau(t)|} R_{i,l,k,m} \qquad (15)$$

8) Critical IP core constraint: One IP core cannot perform multiple operations at each clock cycle, for all $1 \le l \le \lambda$, $1 \le k \le |ven|$, $1 \le m \le |\tau(t)|$

$$\sum_{i=1}^{n} \left(D_{i,l,k,m} + D'_{i,l,k,m} + R_{i,l,k,m}\right) \le 1 \qquad (16)$$

Objective Function: To minimize the purchasing cost for Trojan detection and fast recovery. The objective function can be formulated as follows:

$$\min \sum_{k=1}^{|ven|} \sum_{t=1}^{|\tau|} c(k,t) \times \delta(k,\tau) \qquad (17)$$

5. EXPERIMENTAL RESULTS

The experiments are performed on some known benchmarks. Most of them are chosen from 1992 High-Level Synthesis Benchmarks, and are converted from C language to CDFGs using GAUT tool [8]. They are Polynom with 5 operations, a second-order differential equation solver (diff2) with 11 operations, a DTMF tone generator (dtmf) with 11 operations, a multiple-output second-order filter (mof2) with 12 operations, Ellipticicass with 29 operations and a 16-point finite impulse response with 31 operations. Lingo [13] is used to implement the proposed ILP Formulations. The experiments are performed on a Dell-PC with Intel(R) Core(TM) i5-2400 @3.10GHz CPU and 4.00 GB memory.

5.1 Experimental Results

The ILP may take a very long time to get global optimal results for big benchmarks. We obtain global optimal results for most of the benchmarks in our experiments. Some of the results are not optimal but still is the best result we can get in a short time (within a hour), these results are marked with "*".

Table 3 Designs with Detection Only

Benchmarks	n	λ	A	u	t	v	mc
polynom	5	3	30000	8	6	4	$3580
		6	20000	6	6	5	$3320
diff2	11	4	50000	14	7	5	$4130
		14	30000	9	7	5	$4130
dtmf	11	4	70000	16	5	5	$2960
		8	30000	9	5	5	$2960
mof2	12	7	80000	18	4	4	$2440
		14	40000	8	4	4	$2440
ellipticicass	29	8	30000	28	6	5	$2690
		16	20000	29	7	6	3240*
fir16	31	6	200000	41	5	5	$2960
		12	140000	31	5	5	$2960

Table 4 Designs with Detection and Recovery

Benchmarks	n	λ	A	u	t	v	mc
polynom	5	6	60000	10	9	7	$5140
		12	30000	9	9	6	$5140
diff2	11	8	80000	17	9	7	$5140
		14	30000	9	9	6	$5190
dtmf	11	8	70000	20	6	5	$3830
		15	35000	12	6	5	$3830
mof2	12	14	80000	17	6	5	$3830
		24	40000	22	6	5	$3830
ellipticicass	29	16	50000	31	7	6	$3180*
		24	40000	44	9	8	$4850*
fir16	31	12	220000	39	6	5	$3830
		16	180000	40	6	4	$4390*

In the experiments, we use 8 vendors, each with 3 types of computational IPs which are multipliers, adders and other operators. We set area and cost for types of IP cores from each vendor, which is very similar to the lists shown in Table 1 and is skipped here due to the page limit. The schedule of each benchmark should satisfy the latency and area constraints that are set for them. We compare the results between the designs with detection only (i.e., the designs obtained by [5]) and the designs with detection and recovery. As can be observed from our experimental results, the designs with detection only usually underestimate the need for the diversity of IP cores. And finally, we obtain minimum the purchasing cost and a valid schedule and binding for each benchmark.

Table 3 shows the minimum cost for designs with detection only on each benchmark and Table 4 shows the minimum cost for designs with detection and recovery. Columns n, λ, A, mc represent the number of operations, latency constraint, area constraint, and minimum cost respectively. Columns u, t, v represent u IP cores of t types from v vendors are used in the schedule.

6. CONCLUSION

In this paper, we propose an approach for design with untrusted third party IP cores to support Trojan detection and fast recovery at run time. It exploits diversity of IP cores from various vendors to verify the trustworthiness of IP cores and achieves fast recovery by deactivating Trojan via high-level scheduling and binding. And we show how to obtain the minimal cost for a set of given design constraints and a list of IP core vendors using ILP method. The experimental results show that the existing detection-only technique often underestimates the actual need for the diversity of IP cores.

7. REFERENCES

[1] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, S. Bhunia, "MERO: A Statistical Approach for Hardware Trojan Detection", *Cryptographic Hardware and Embedded Systems (CHES)*, 2009.

[2] S. Jha and S.K. Jha, "Randomization Based Probabilistic Approach to Detect Trojan Circuits", *11th IEEE High Assurance Systems Engineering Symposium*, 2008.

[3] Y. Jin and Y. Makris, "Hardware Trojan Detection using Path Delay Fingerprint," *HOST*, 2008.

[4] R. M. Rad, J. Plusquellic and M. Tehranipoor, "Sensitivity Analysis to Hardware Trojan using Power Supply Transient Signals," *HOST08.*

[5] J. Rajendran, H. Zhang, O. Sinanoglu, R. Karri, "High-level Synthesis for Security and Trust", On-Line Testing Symposium (IOLTS), 2013.

[6] R.S. Chakraborty, S. Narasimhan, S. Bhunia, "Hardware Trojan: Threats and emerging solutions", *High Level Design Validation and Test Workshop, 2009. HLDVT 2009. IEEE International* , vol., no., pp.166,171, 4-6 Nov. 2009.

[7] S. Bhunia, M. Abramovici, D. Agrawal, P. Bradley, M.S. Hsiao, J. Plusquellic, M. Tehranipoor, "Protection Against Hardware Trojan Attacks: Towards a Comprehensive Solution", *Design & Test, IEEE* , vol.30, no.3, pp.6,17, June 2013.

[8] E. Martin, O. Sentieys, H. Dubois, J-L. Philippe, "GAUT: An architectural synthesis tool for dedicated signal processors", *Design Automation Conference, 1993, with EURO-VHDL '93. Proceedings EURO-DAC '93., European* , pp.14,19, 20-24 Sep 1993.

[9] T. Karnik, P. Hazucha, "Characterization of Soft Errors Caused by Single Event Upsets in CMOS Process", *IEEE Transactions on Dependable and Secure Computing*, June, 2004.

[10] R. Gaillard, "Single Event Effects: Mechanisms and Classification", Soft Errors in Modern Electronic Systems Frontiers in Electronic Testing Vol 41, pp 27-54, 2001.

[11] I. Hwang, S. Kim, Y. Kim, C.E. Seah, "A Survey of Fault Detection, Isolation and Reconfiguration Methods", IEEE Transactions on Control Systems Technology, Vol. 18, pp. 636-653, May, 2010.

[12] M. E. Orchard, G. J. Vachtsevanos, "A Particle-filtering Approach for On-line Fault Diagnosis and Failure Prognosis", Transactions of the Institute of Measurement and Control, Vol. 31, pp. 221-246, August, 2009.

[13] Lingo, available online at www.lindo.com.