HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection

Rajat Subhra Chakraborty, *Student Member, IEEE*, and Swarup Bhunia, *Senior Member, IEEE*

***Abstract*—Hardware intellectual-property (IP) cores have emerged as an integral part of modern system-on-chip (SoC) designs. However, IP vendors are facing major challenges to protect hardware IPs from IP piracy. This paper proposes a novel design methodology for hardware IP protection using netlist-level obfuscation. The proposed methodology can be integrated in the SoC design and manufacturing flow to simultaneously obfuscate and authenticate the design. Simulation results for a set of ISCAS-89 benchmark circuits and the advanced-encryption-standard IP core show that high levels of security can be achieved at less than 5% area and power overhead under delay constraint.**

***Index Terms*—Design for security, hardware authentication, hardware obfuscation, *intellectual-property* (IP) piracy, IP protection.**

I. Introduction

REUSE-BASED system-on-chip (SoC) design using hardware *intellectual-property* (IP) cores has become a pervasive practice in the industry. The IP cores usually come in the form of synthesizable register-transfer-level (RTL) descriptions ("Soft IP"), gate-level designs directly implementable in hardware ("Firm IP"), or GDS-II design database ("Hard IP"). The approach of designing complex systems by integrating tested, verified, and reusable modules reduces the design time and cost dramatically [1].

Unfortunately, recent trends in IP-piracy and reverse-engineering efforts to produce counterfeit ICs have raised serious concerns in the IC design community [1]–[4]. IP piracy can take several forms, as illustrated by the following scenarios.

1) A chip design house buys an IP core from an IP vendor and makes an illegal copy or "clone" of the IP. The IC design house then sells it to another chip design house (after minor modifications) claiming the IP to be its own [2].
2) An untrusted fabrication house makes an illegal copy of the GDS-II database supplied by a chip design house and then illegally sells them as hard IP.
3) An untrusted foundry manufactures and sells counterfeit copies of the IC under a different brand name [3].
4) An adversary performs postsilicon reverse engineering on an IC to manufacture its illegal clone [4].

Manuscript received August 27, 2008; revised January 30, 2009. Current version published September 18, 2009. This paper was recommended by Associate Editor V. Narayanan.

The authors are with the Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, OH 44106 USA (e-mail: rsc22@case.edu; skb21@case.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2009.2028166

These scenarios demonstrate that all parties involved in the IC design flow are vulnerable to different forms of IP infringement which can result in loss of revenue and market share. Hence, there is a critical need of a piracy-proof design flow that equally benefits the IP vendor, the chip designer, as well as the system designer. A desirable characteristic of such a secure design flow is that it should be *transparent* to the end-user, i.e., it should not impose any constraint on the end-user with regard to its usage, cost, or performance.

Existing approaches for hardware protection do not ensure comprehensive protection to IP vendors, chip designers, system designers, and end-users [1]–[4], [6]–[10]. They are targeted toward preserving the rights of only *a single* party (mostly IP vendors). In order to benefit all of them simultaneously, we require an *antipiracy design flow* in which IP vendors, designers, and manufacturers take an active part in securing their own rights and ensure that the customer gets an authentic and trustworthy product. Furthermore, existing works in this field primarily focus on protecting the hardware from illegal copy. They do not address the issue of achieving resistance against reverse engineering during fabrication, test, and deployment.

Obfuscation is a technique that transforms an application or a design into one that is functionally equivalent to the original but is significantly more difficult to reverse engineer [5]. In this paper, we present ***HARPOON***, a SoC design methodology for **HAR**dware **P**rotection through **O**bfuscation **O**f **N**etlist. The proposed obfuscation-based protection scheme provides antipiracy and tamperproof qualities to the hardware at every stage of the hardware design and manufacturing. In particular, we make the following major contributions.

1) We present a low-overhead and low-complexity IP design methodology for gate-level IPs aimed toward preventing IP piracy. The proposed design methodology provides simultaneous *obfuscation* and *authentication* of a netlist, thus protecting valuable hardware IPs. The key step to accomplish the obfuscation goal is to systematically modify the state-transition function and internal-circuit structure in such a way that the circuit operates in normal mode only upon application of a predefined enabling sequence of patterns at primary inputs, referred to as "key" for the circuit.
2) We provide a metric to quantify the level of obfuscation and present theoretical analysis to evaluate the effect of design modifications on resulting obfuscation. Such an analysis is used to explore techniques for achieving maximum obfuscation at minimal design overhead.

3) We present a design flow for SoCs based on the proposed hardware IP obfuscation technique. To the best of our knowledge, this is the first gate-level obfuscation-based design flow which can provide security at multiple stages of the IC life cycle.

The rest of this paper is organized as follows. In Section II, we discuss previous work in this field and the motivation behind this paper. In Section III, we present an obfuscation metric and provide theoretical analysis to evaluate an obfuscation scheme. In Section IV, we describe the proposed obfuscation scheme and develop a design methodology to integrate it in the SoC design and manufacturing flow. In Section V, we present simulation results for a set of benchmark circuits and the *advanced-encryption-standard* (AES) encryption/decryption IP core. We conclude in Section VI.

II. Related Work

Hardware IP protection has been investigated earlier in diverse contexts, addressing *licensed* as well as the prelicense *evaluation version* of an IP. Previous work on IP protection can be broadly classified into two main categories: 1) *obfuscation-based* protection and 2) *authentication-based* protection.

A. Obfuscation-Based IP Protection

In obfuscation-based IP protection, the IP vendor usually affects the human readability of the hardware-description-language (HDL) code [6] or relies on cryptographic techniques to encrypt the source code [7]–[10]. In [6], the code is reformatted by changing the internal net names and removing the comments, so that the circuit description is no longer intelligible to the human reader. However, it does not modify the functionality of the IP core, and thus cannot prevent it from being stolen by a hacker and used as a "black-box" module. In a more common practice widely adopted in the industry [7], [10], the HDL source code is encrypted, and the IP vendor provides the key to decrypt the source code only to its valid customers. However, this technique may enforce the use of a particular design platform [8], [9], a situation that might be unacceptable to many SoC designers who seek the flexibility of multiple tools from diverse vendors in the design flow. Moreover, none of these techniques prevent possible reverse-engineering effort at later stages of the design and manufacturing flow.

B. Authentication-Based IP Protection

To protect the rights of the IP vendor through authentication, the approaches proposed are directed toward embedding a *digital watermark* in the design [1], [2], which helps to authenticate the design at a later stage. Typically, this is inserted by design modifications which result in one or multiple input–output response pair(s) which do not arise during the normal functioning of the IP. Such digital signatures are known only to the IP vendor. Since this digital watermark (or signature) cannot be removed from the IP, it helps to prove an illegal use of such a component in litigation. However, the effectiveness of authentication-based IP protection schemes is limited by the fact that these techniques are *passive*, and hence, they cannot prevent the stolen IP from being used.

The approaches directed toward preventing the rights of the IC designer, on the other hand, ensure that the design house has a knowledge of *every* IC instance manufactured and sold in the market. Usually, this is implemented by including a *locking* mechanism in the IC [11], [12]. The fabrication facility would require a unique bit sequence provided by the design house to "unlock" the IC. This unique bit sequence is determined by a *physically unclonable function* (PUF) block that is added to the design by the design house. However, such approaches cannot prevent the possibility of reverse engineering a design to expose its functionality as well as the security scheme. Moreover, they do not address protecting the right of an IP vendor.

III. Analysis of Netlist Obfuscation

In order to achieve comprehensive protection of hardware IPs, the proposed approach focuses on obfuscating the functionality and structure of an IP core by modifying the gate-level netlist, such that it both obfuscates the design and embeds authentication features in it. The IC is protected from unauthorized manufacturing by the fact that the system designer depends on input from the chip designer to use the IC. Consequently, the manufacturing house cannot simply manufacture and sell unauthorized copies of an IC without the knowledge of the design house. In addition, by adopting a PUF-based activation scheme, the security can be increased further, since it ensures that the activation pattern is specific to each IC instance. The embedded authentication feature helps in proving illegal usage of an IP during litigation. Finally, the obfuscation remains transparent to the end-user who has the assurance of using a product that has gone through an antipiracy secure design flow.

The obfuscation is carried out in a manner that, for a target design overhead, achieves maximum resistance to any reverse-engineering effort, whether structural or functional. We do not rule out the possibility that a sophisticated adversary can have access to superior computational capabilities and powerful computer-aided-design tools. Hence, it is important to mathematically analyze the level of obfuscation and ensure that it is practically infeasible for an adversary to reverse engineer an obfuscated design.

A. Hardware IP Piracy: The Hacker's Perspective

A hacker trying to determine the functionality of an obfuscated gate-level IP core can take resort to either of the following ways: 1) simulation-based reverse engineering to determine functionality of the design or 2) structural analysis of the netlist to identify and isolate the original design from the obfuscated design. The proposed obfuscation approach targets to achieve simulation mismatch for the maximum possible input vectors, as well as structural mismatch for maximum possible circuit nodes. To achieve structural mismatch between the reference and the obfuscated design, we modify the state-transition function as well as internal logic structure.

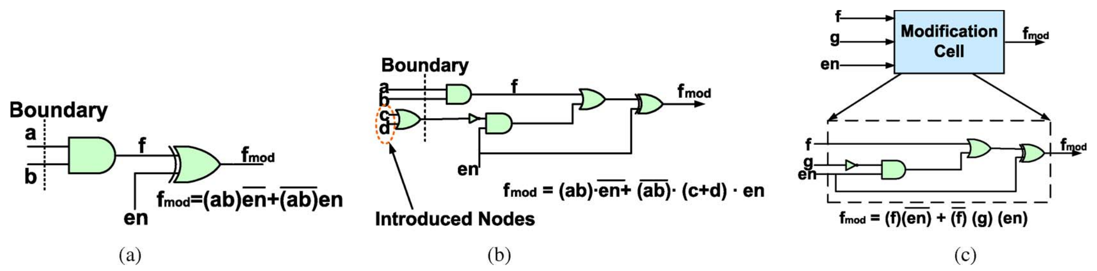

Boundary
a
b
f
en
f_{mod}
$f_{mod}=(ab)\overline{en}+\overline{(ab)}en$
Boundary
a
b
c
d
f
en
f_{mod}
$f_{mod} = (ab)\cdot\overline{en} + \overline{(ab)}\cdot(c+d)\cdot en$
Introduced Nodes
f
g
en
Modification Cell
f_{mod}
$f_{mod} = (f)(\overline{en}) + (\overline{f})(g)(en)$
(a)
(b)
(c)

Fig. 1. Schemes for Boolean function modification and modification cell. (a) Simple scheme using XOR gate. (b) Scheme using expansion of logic cone. (c) Modification cell.

1) Modification scheme employing input logic-cone expansion: Consider the simple example shown in Fig. 1(a). It shows a modified two-input AND gate. If $en = 0$, it works as an ordinary AND gate; however, if $en = 1$, the original functionality of the AND gate is obfuscated because the output is inverted. Simulation of the simple circuit shown in Fig. 1(a) against an ordinary two-input AND gate will report four possible input vectors with $en = 1$ as failing patterns. To increase the number of failing patterns for this circuit, we must increase its input logic-cone size, while ensuring that it continues to function properly when $en = 0$. Fig. 1(b) shows an alternative scheme, where the input logic cone has been expanded to include the node c and d. A complete enumeration of the truth table of the modified circuit will show failures for 13 input patterns.

The modification scheme shown in Fig. 1(b) can be generalized to a form shown in Fig. 1(c). Here, f is the Boolean function corresponding to an internal node, and g is any arbitrary Boolean logic function. It is worthwhile to note that the simple modification scheme shown in Fig. 1(a) is a special case with $g = 1$. As shown, the modified logic function is of the form

$$f_{\text{mod}} = f \cdot \overline{en} + \overline{f} \cdot g \cdot en. \tag{1}$$

Let us call the function g as the *modification kernel function* (**MKF**). It is clear that, for $en = 1$, if $g = 1$ for a given set of primary inputs and state-element (SE) output state, $f_{\text{mod}} = \overline{f}$, and the test pattern is a failing test pattern. To increase the amount of dissimilarity between the original and modified designs, we should try to make g evaluate to logic-1 as often as possible. At first glance, the trivial choice seems to be $g = 1$. However, in that case, the input logic cone is not expanded, and thus, the number of failing vectors reported by a formal verification approach is limited. For any given set of inputs, this is achieved by a logic function which is the logical OR of the input variables.

B. Obfuscation Metric

We now derive a metric that quantifies the level of obfuscation, i.e., the mismatch between the original and the obfuscated design. Let f be a function of the set P_1 of primary inputs and SE outputs and g be a function of a set P_2 of primary inputs and SE outputs. Let $P_1 \bigcap P_2 = P$, $|P_1| = p_1$, $|P_2| = p_2$, $|P| = p$, $P_1 \bigcup P_2 = \Gamma$, and $|\Gamma| = \gamma = p_1 + p_2 - p$. Furthermore, let g be a Boolean OR function with p_2 inputs. Then, for $(2^{p_2} - 1)$ of its input combinations, g is at logic-1. Consider $en = 1$. Then, for all these $(2^{p_2} - 1)$ input combinations of P_2, $f_{\text{mod}} = \overline{f}$, causing a failing vector. Corresponding to each of these $(2^{p_2} - 1)$ combinations of P_2, there are $(p_1 - p)$ other independent primary inputs to f. Hence, the total number of failing vectors when $g = 1$ is

$$N_{g1} = 2^{(p_1 - p)} \cdot (2^{p_2} - 1). \tag{2}$$

For the other "all-zero" input combination of P_2, $f = 0$. Let the number of possible cases where $f = 1$ at $g = 0$ be N_{g0}. Then, the total number of failing input patterns

$$N_{\text{failing}} = N_{g1} + N_{g0} = 2^{(p_1 - p)} \cdot (2^{p_2} - 1) + N_{g0}. \tag{3}$$

In the special case when $P_1 \bigcap P_2 = P = \phi$, N_{g0} is given simply by the number of possible logic-1 entries in the truth table of f.

The total input space of the modified function has a size 2^γ. We define the *obfuscation metric* (M) as

$$M = \frac{N_{\text{failing}}}{2^{\gamma+1}} = \frac{2^{(p_1 - p)} \cdot (2^{p_2} - 1) + N_{g0}}{2^{p_1 + p_2 - p + 1}}. \tag{4}$$

The "+1" factor in the denominator is due to the en signal. Note that $0 < M \leq (1/2)$. As an example, for $f = ab + cd$, with $g = a + b$, $M = 13/32$. As a special case, consider $\mathbf{p} = \mathbf{p_1} = \mathbf{p_2}$, i.e., the signal g is derived from the same set of primary inputs of f. Then

$$M = \frac{1}{2}\left(1 + \frac{N_{g0} - 1}{2^{p_1}}\right) = \frac{1}{2}\left(1 + \frac{N_{g0} - 1}{2^{p_2}}\right). \tag{5}$$

In this case, N_{g0} is either zero or one; hence, $M = 1/2$ if $N_{g0} = 1$ and $M = (1/2)(1 - 2^{-p_1})$ if $N_{g0} = 0$. Note that M attains the maximum (ideal) value of 0.5 in this case when $N_{g0} = 1$. The theoretical maximum, however, is not a very desirable option because it keeps the input space limited to 2^{p_1} possible vectors. Again, if $\mathbf{p} = \mathbf{0}$, i.e., g is generated by a completely different set of primary inputs which were not included in f, then

$$M = \frac{1}{2}\left(1 + \frac{N_{g0} 2^{-p_1} - 1}{2^{p_2}}\right). \tag{6}$$

Larger values of N_{g0} and smaller values of p_2 for a given p_1 help to increase M. Note that, unlike the first case, N_{g0} is guaranteed to be nonzero. This property effectively increases the value of M in case 2) than that in case 1) for most functions. However, in the second case, $M < 1/2$, i.e., M cannot attain the theoretical maximum value.

Selection of the MKF (**g**)*:* Although the earlier analysis points to the selection of primary inputs or SE outputs to design the **MKF** g satisfying the condition $p = 0$, in practice, this could incur a lot of hardware overhead to generate the OR functions corresponding to each modified node. An alternative approach is to select an internal logic node of the netlist to provide the Boolean function g. It should have the following characteristics.

1) The modifying node should have a very large fan-in cone, which, in turn, would substantially expand the logic cone of the modified node.
2) It should not be in the fan-out cone of the modified node.
3) It should not have any node in its fan-in cone which is in the fan-out cone of the modified node.

Conditions (2) and (3) are essential to prevent any *combinational loop* in the modified netlist. Such a choice of g does not, however, guarantee it to be an OR function and is thus suboptimal. We explore the effectiveness of this MKF in Section V.

IV. System-Level Obfuscation Methodology

In this section, we present the secure SoC design methodology for hardware protection based on the analysis presented in the previous section.

A. State-Transition Function Modification

The first step of the obfuscation procedure is the modification of the state-transition function of a sequential circuit by inserting a small finite state machine (FSM). The inserted FSM has all or a subset of the primary inputs of the circuit as its inputs (including the clock and reset signals) and has multiple outputs. At the start of operations, the FSM is reset to its initial state, forcing the circuit to be in the *obfuscated* mode. Depending on the applied input sequence, the FSM then goes through a state-transition sequence and only on receiving N specific input patterns in sequence goes to a state which lets the circuit operate in its *normal* mode. The initial state and the states reached by the FSM before a successful initialization constitute the "preinitialization state space" of the FSM, while those reached after the circuit has entered its normal mode of operation constitute the "postinitialization state space." Fig. 2 shows the state diagram of such an FSM, with $P0 \rightarrow P1 \rightarrow P2$ being the correct initialization sequence. The input sequence $P0$ through $P2$ is decided by the IP designer.

The FSM controls the mode of circuit operation. It also modifies selected nodes in the design using its outputs and the *modification cell* (e.g., M_1 through M_3). This scheme is shown in Fig. 2 for a gate-level design that incorporates modifications of three nodes n_1 through n_3. The **MKF** can either be a high-fan-in internal node (avoiding combinational loops) in the unmodified design or the OR function of several selected primary inputs. The other input (corresponding to the en port of the modification cell) is a Boolean function of the inserted FSM state bits with the constraint that it is at logic-0 in the *normal* mode. This modification ensures that, when the

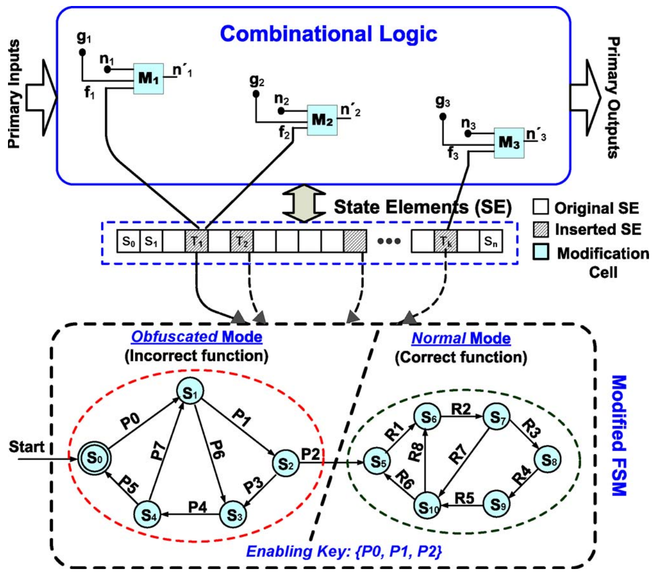


Fig. 2. Proposed functional and structural obfuscation scheme by modification of the state-transition function and internal node structure.

FSM output is at logic-0, the logic values at the modified nodes are the same as the original ones. On the other hand, in the *obfuscated* mode, for any FSM output that is at logic-1, the logic values at the modified nodes are inverted if $g = 1$ and logic-0 if $g = 0$. Provided the modified nodes are selected judiciously, modifications at even a small number of nodes can greatly affect the behavior of the modified system. This happens even if the en signal is not always at logic-0. In our implementation, we chose to have the number of outputs of the inserted FSM as a user-specified parameter. These outputs are generated as random Boolean functions of the SE bits at design time with the added constraint that, in the *normal* mode, they are at logic-0. The randomness of the Boolean functions adds to the security of the scheme. Such a node modification scheme can provide higher resistance to structural reverse-engineering efforts than the scheme in [14].

B. Embedding Authentication Features

The proposed obfuscation scheme allows us to easily embed authentication signature into a gate-level design with negligible design overhead. Such an embedded signature acts as a *digital watermark* and hence helps to prevent attack from trusted parties in the design flow with knowledge of initialization sequence. Corresponding to each state in the *preinitialization state space*, we arrange to have a particular pattern to appear at a subset of the primary outputs when a predefined input sequence is applied. Even if a hacker arranges to bypass the initialization stage by structural modifications, the inserted FSM can be controlled to have the desired bit patterns corresponding to the states in the *preinitialization state space*, thus revealing the watermark. For postsilicon authentication, scan flip-flops can be used to bring the design to the obfuscated mode. Fig. 3 shows the modification of the state-transition function for embedding authentication signature in the *obfuscated mode* of operation.

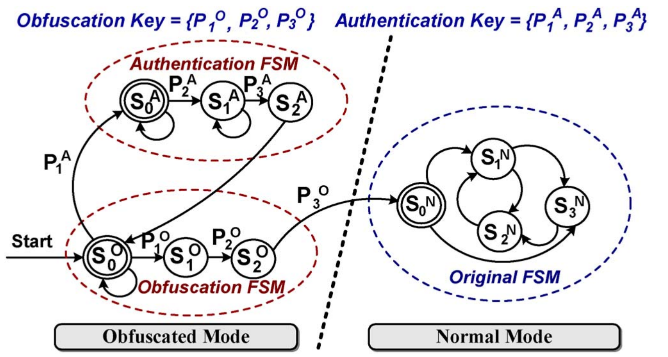


Fig. 3. Modification of the initialization state space to embed authentication signature.

To mask or disable the embedded signature, a hacker needs to perform the following steps, assuming a purely random approach.

1) Choose the correct inserted FSM SEs (n_p) from all the total SEs (n_t). This has $\binom{n_t}{n_p}$ possible choices.
2) Apply the correct input vector at the n_i input ports where the vectors are to be applied to get the signature at the selected n_o output ports. This is one out of 2^{n_i} choices.
3) Choose the n_o primary outputs at which the signature appears from the total set of primary outputs (n_{po}). This has $\binom{n_{po}}{n_o}$ possibilities.
4) For each of these recognized n_o outputs, identify it to be one among the possible $2^{2^{(n_i+n_p)}}$ Boolean functions (in the *obfuscated* mode) of the n_i primary inputs and n_p SEs and change it without changing the normal functionality of the IP.

Hence, in order to mask one signature, the attacker has to make exactly one correct choice from among $N = \binom{n_t}{n_p} \cdot 2^{n_i} \cdot \binom{n_{po}}{n_o} \cdot 2^{2^{(n_i+n_p)}}$ possible choices, resulting in a masking success probability of $P_{\text{masking}} \cong 1/N$. To appreciate the scale of the challenge, consider a case with $n_t = 30$, $n_p = 3$, $n_i = 4$, $n_o = 4$, and $n_{po} = 16$. Then, $P_{\text{masking}} \sim 10^{-47}$. In actual IPs, the masking probability would be substantially lower because of higher values of n_p and n_t.

C. Choice of Optimal Set of Nodes for Modification

To obfuscate a design, we need to choose an optimal set of nodes to be modified, so that maximum obfuscation is achieved under the given constraints. We estimate the level of obfuscation by the amount of verification mismatch reported by a formal-verification-based equivalence checker tool. Formal equivalence checker tools essentially try to match the input logic cones at the SEs and the primary outputs of the reference and the implementation [15]. Hence, nodes with larger fan-out logic cone would be preferred for modification since that will, in turn, affect the input logic of comparatively larger number of nodes. In addition, large input logic cone of a node is generally indicative of its higher *logic depth*; hence, any change at such a node is likely to alter a large number of primary outputs. Thus, in determining the suitability metric for a node as a candidate for modification, we need to consider both these factors. We

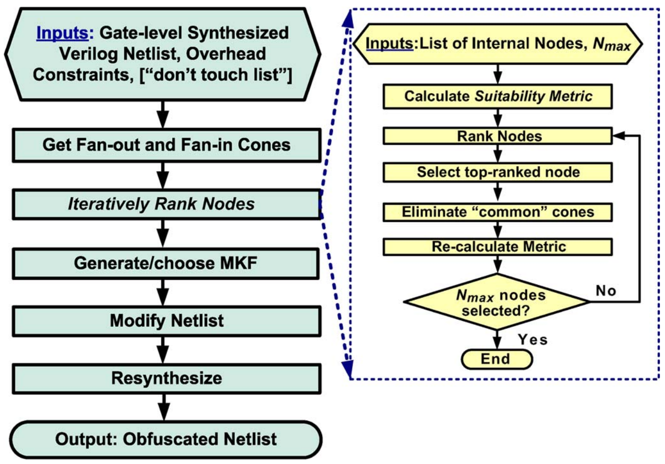

Inputs: Gate-level Synthesized Verilog Netlist, Overhead Constraints, ["don't touch list"]
Get Fan-out and Fan-in Cones
Iteratively Rank Nodes
Generate/choose MKF
Modify Netlist
Resynthesize
Output: Obfuscated Netlist
Inputs:List of Internal Nodes, Nmax
Calculate Suitability Metric
Rank Nodes
Select top-ranked node
Eliminate "common" cones
Re-calculate Metric
Nmax nodes selected?
No
Yes
End

Fig. 4. Hardware obfuscation design flow along with steps of the iterative node ranking algorithm.

propose the following metric to be used as the *suitability metric* for a node:

$$M_{\text{node}} = \left(\frac{w_1 \cdot FO}{FO_{\max}} + \frac{w_2 \cdot FI}{FI_{\max}}\right) \times \frac{FO \cdot FI}{FI_{\max} \cdot FO_{\max}} \quad (7)$$

where FI and FO are the number of nodes in the fan-in and the fan-out cones of the node, respectively. $FI_{\max}$ and $FO_{\max}$ are the maximum number of fan-in and fan-out nodes in the circuit netlist and are used to normalize the metric. w_1 and w_2 are weights assigned to the two factors, with $0 \le w_1$, $w_2 \le 1$, and $w_1 + w_2 = 1$. We chose $w_1 = w_2 = 0.5$, which gives the best results in terms of obfuscation, as shown in the next section. Note that $0 < M_{\text{node}} \le 1$. Because of the widely differing values of $FO_{\max}$ and $FI_{\max}$ in some circuits, it is important to consider both the sum and the product terms involving $FO/FO_{\max}$ and $FI/FI_{\max}$. Considering only the sum or the product term results in an inferior metric that fails to capture the actual suitability of a node, as observed in our simulations.

D. HARPOON Design Methodology

The overall hardware obfuscation design flow is shown in Fig. 4. First, from the synthesized gate-level HDL netlist of an IP core, the fan-in and fan-out cones of the nodes are obtained. Then, an iterative ranking algorithm is applied to find the most suitable $N_{\max}$ modifiable nodes, where $N_{\max}$ is the maximum number of nodes that can be modified within the allowable overhead constraints. The ranking is a multipass algorithm, with the metric for each node being dynamically modified based on the selection of the node in the last iteration. The algorithm takes into account the overlap of the fan-out cones of the nodes which have been already selected and eliminates them from the fan-out cones of the remaining nodes. On the completion of each iteration, the top-ranking node among the remaining nodes is selected, so that selection of $N_{\max}$ nodes would take $N_{\max}$ iterations. In this way, as the iterations progress, the nodes with more nonoverlapping fan-out cones are assigned higher weight. We observed the superiority of this iterative approach over a single-pass ranking approach for all the benchmark circuits we considered.

A "*do not touch*" list of nodes can be optionally input to direct it not to modify certain nodes, e.g., nodes which fall in the critical path. In large benchmarks, we observed that there were sufficient nodes with high fan-outs, such that skipping a few "*do not touch*" nodes still maintains the effectiveness of node-modification algorithm in achieving functional and structural obfuscation. For each node to be modified, proper **MKF** (g) is selected either on the basis of its fan-in cone size or by OR-ing several primary inputs which were originally not present in its input logic cone. The FSM is then integrated with the gate-level netlist, and the selected nodes are modified. The modified design is resynthesized and flattened to generate a new gate-level netlist. The integrated FSM and the modification cells are no longer visually identifiable in the resultant netlist. This resynthesis is performed under timing constraint, so that it maintains circuit performance.

The IP vendor applies the hardware obfuscation scheme to create a modified IP and supplies it to the design house, along with the activating sequences. The design house receives one or multiple IPs from the IP vendors and integrates them on chip. To activate the different IPs, the designer needs to include a low-overhead *controller* in the SoC. This controller module can perform the initialization of the different IP blocks in two different ways. In the first approach, it serially steers the different initialization sequences to the different IP blocks from the primary inputs. This controller module will include an integrated FSM which determines the steering of the correct input sequences to a specific IP block. Multiplexors are used to steer initialization sequences to the IP blocks or the primary inputs and internal signals during normal operation. The chip designer must modify the test benches accordingly to perform block-level or chip-level logic simulations.

In the second approach, the initialization sequences is stored permanently on-chip in a read-only memory (ROM). In the beginning of operations, the controller module simply reads the different input sequences in parallel and sends them to the different IP blocks for initialization. The advantage of this approach is that the number of initialization cycles can be limited. However, additional overhead is incurred for storing the input sequences in an on-chip ROM. To increase the security of the scheme, the chip designer can arrange an instance-specific initialization sequence to be stored in a one-time programmable ROM. In that case, following the approach in [11], we can have the activating patterns to be simple logic function (e.g., and XOR) of the patterns read from the ROM and the output of a **PUF** block. The patterns are written to the ROM postmanufacturing after receiving instructions from the chip designer, as suggested in [11]. Because the output of a PUF circuit is not predictable before manufacturing, it is not possible to have the same bits written into the programmable ROMs for each IC instance. Fig. 5 shows this scheme.

The manufacturing house manufactures the SoC from the design provided by the design house and passes it on to the test facility. If a PUF block has been used in the IC, the test engineer reports the output on the application of certain vectors back to the chip designer. The chip designer then calculates the

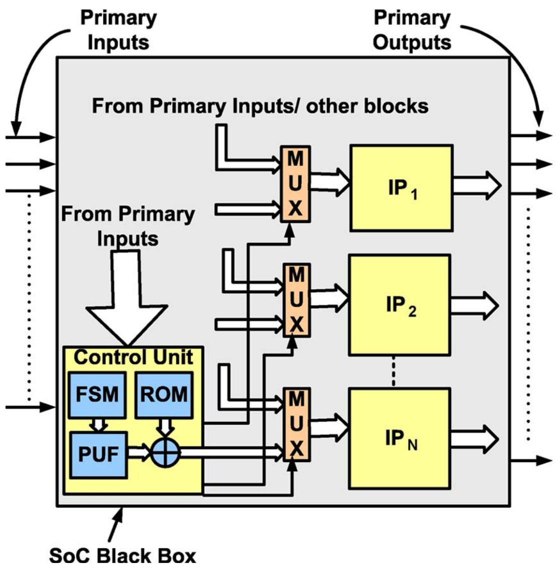


Fig. 5. SoC design modification to support hardware obfuscation. An on-chip controller combines the input patterns with the output of a PUF block to produce the activation patterns.

specific bits required to be written in the one-time programmable ROM. The test engineer does so and blows off a one-time programmable fuse, so that the output of the PUF block is no longer visible at the output. The test engineer then performs postmanufacturing testing, using the set of test vectors provided by the design house. Ideally, all communications between parties associated with the design flow should be carried out in an encrypted form, using symmetric or asymmetric cryptographic algorithms such as Diffie–Hellman [12]. The tested ICs are passed to the system designer along with the initialization sequence (again in an encrypted form) from the design house.

The system designer integrates the different ICs in the board-level design and arranges to apply the initialization patterns during "booting" or similar other initialization phase. Thus, the initialization patterns for the different SoCs need to be stored in ROM. In most application-specific integrated circuits composed of multiple IPs, several initialization cycles are typically needed at start-up to get into the "steady-stream" state, which requires accomplishing certain tasks such as the initialization of specific registers [16]. The system designer can easily utilize this inherent latency to hide the additional cycles due to initialization sequences from the end-user.

Finally, this secure system is used in the product for which it is meant. It provides the end-user with the assurance that the components have gone through a secure and piracy-proof design flow. Fig. 6 shows the challenges and benefits of the design flow from the perspectives of different parties associated with the flow. It is worth noting that the proposed design methodology remains valid for a SoC design house that uses custom logic blocks instead of reusable IPs. In this case, the designer can synthesize the constituent logic blocks using the proposed obfuscation methodology for protecting the SoC.

V. Results

In this section, we present simulation results to show the effectiveness of the proposed hardware obfuscation methodology

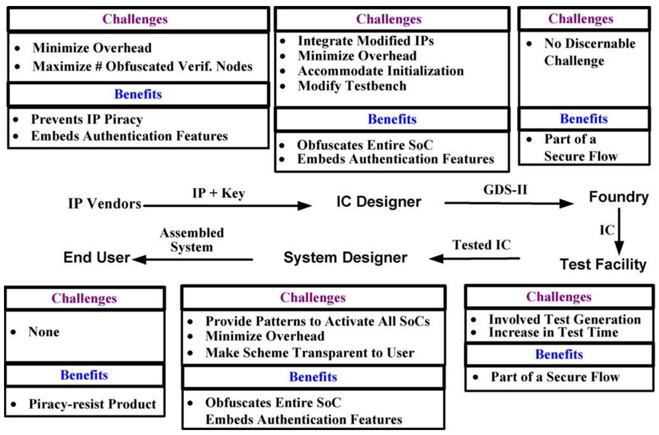


Fig. 6. Challenges and benefits of the *HARPOON* design methodology at different stages of a hardware IP life cycle.

TABLE I
AVERAGE NUMBER OF FAILING PATTERNS FOR ISCAS-89 BENCHMARK CIRCUITS FOR DIFFERENT MODIFICATION SCHEMES

Benchmark	Scheme-1	Scheme-2	Scheme-3
s298	51	158	193
s344	215	1093	1233
s444	197	569	772
s526	146	485	1186
s641	598	2491	5135
s713	913	2928	3301
s838	382	1757	5106
s1196	2423	5382	9573
s1238	2552	5157	9511
s1423	6431	18120	28350
s1488	333	1816	1156
s5378	13311	29482	53066
s9234	13862	30385	53365

TABLE II
NUMBER OF FAILING PATTERNS FOR ISCAS-89 BENCHMARK CIRCUITS

Benchmark Circuit	Area Constraint			
	5%	10%	15%	20%
s298	176	184	202	210
s344	1010	1290	1228	1402
s444	364	638	1004	1082
s526	416	1216	1270	1842
s641	3723	3185	6045	7585
s713	3084	3143	3210	3765
s838	1742	5118	5466	8096
s1196	9631	10321	8931	9408
s1238	8442	9876	9780	9944
s1423	11868	25263	29007	47262
s1488	760	1096	1344	1422
s5378	48799	52059	55961	56638
s9234	13862	30385	53365	56638

for a set of ISCAS-89 benchmark circuits [17], as well as the AES encrypter/decrypter IP core [18].

A. Simulation Setup

All circuits were synthesized using Synopsys *Design Compiler* with optimization parameters set for minimum area and mapped to a LEDA 250-nm standard cell library. The flow was developed using the TCL scripting language and was directly integrated in the *Design Compiler* environment. All

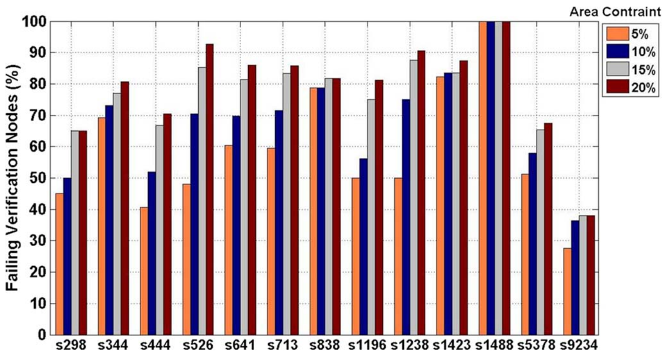


Fig. 7. Observed verification failures (with application of the *HARPOON* methodology) for ISCAS-89 circuits.

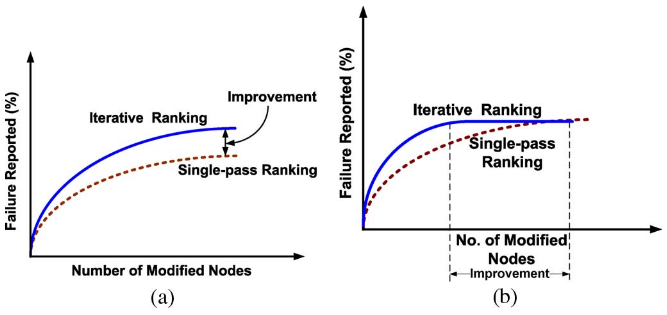


Fig. 8. Improvement with the iterative node ranking algorithm. (a) Verification failure percentage and (b) number of nodes to be modified at isofailure percentage.

formal verifications were carried out using Synopsys *Formality*. The verification nodes considered by Formality constituted of the inputs of SEs (e.g., flip-flops) and primary outputs.

B. Results for ISCAS-89 Benchmark Circuits

Choice of scheme: A simple four-state FSM was designed for each of the benchmarks to achieve hardware and functional obfuscation. We initially explored three choices—the simple node modification scheme using only XOR gates (scheme 1), the theoretically suggested modification scheme employing OR-ing of selected primary inputs (scheme 2), and lastly, the low-overhead modification scheme employing random selection of internal nodes and avoiding combinational loops (scheme 3). Then, the maximum number of modifiable nodes $N_{\max}$ for each benchmark circuit was determined considering four different area constraints (5%, 10%, 15%, and 20%). For all the benchmarks, the number of nodes modified was less than 5% of the total number of nodes in the netlist. Table I shows the number of total *failing patterns* reported by *Formality* for the benchmarks, averaged over the different area constraints. From these results, it is clear that the random-node-selection algorithm (avoiding combinational loops) achieves the highest number of failing vectors for a given area overhead constraint. Hence, in the rest of this section, all the results presented would be for scheme-3.

Obfuscation effects: The benchmarks were then subjected to the hardware obfuscation design flow (including the iterative ranking algorithm). Table II shows the number of failing patterns for the ISCAS-89 benchmark circuits for different projected area overheads. The modified and resynthesized

TABLE III
DESIGN OVERHEADS (IN PERCENT) FOR DIFFERENT AREA CONSTRAINTS

Benchmark Circuit	5% area constraint			10% area constraint			15% area constraint			20% area constraint		
	Area	**Delay**	**Power**	**Area**	**Delay**	**Power**	**Area**	**Delay**	**Power**	**Area**	**Delay**	**Power**
s298	3.91	0.00	5.26	8.64	0.00	14.04	14.74	0.50	16.67	19.29	0.00	18.42
s344	3.45	-2.98	5.38	8.83	-8.96	9.87	14.89	-6.20	14.35	18.59	-7.90	18.83
s444	4.63	0.00	9.62	9.50	0.00	18.27	14.15	0.00	20.19	18.26	0.00	23.08
s526	3.64	0.00	7.06	8.12	0.00	16.17	14.89	-1.60	21.87	19.30	-0.78	25.28
s641	4.96	-3.66	8.20	9.74	-3.66	13.90	14.02	-4.60	19.59	19.63	-4.20	23.01
s713	4.06	-3.66	7.34	9.07	-3.66	14.69	14.43	-2.60	20.34	19.26	-2.60	22.88
s838	2.20	-2.39	6.92	9.00	-8.57	9.76	14.17	-5.50	12.93	19.13	0.00	14.00
s1196	3.96	0.00	6.04	6.06	0.00	16.09	14.45	-1.76	19.14	18.10	0.00	20.55
s1238	4.52	-0.42	6.52	9.99	-0.42	9.97	14.87	0.00	18.26	18.23	-0.90	23.79
s1423	4.70	-0.78	8.02	9.61	-2.64	14.91	14.97	-1.08	22.43	19.99	-2.42	26.19
s1488	3.27	-2.79	3.13	8.65	-0.93	8.33	13.19	0.00	10.49	18.17	0.00	13.62
s5378	4.34	0.00	8.91	9.87	0.00	13.80	13.84	0.00	20.43	19.93	0.00	23.70
s9234	4.74	0.00	5.80	8.82	3.60	12.37	8.82	3.50	15.52	14.29	3.80	19.72
Average	4.03	-1.28	8.83	8.92	-1.94	13.31	14.38	-1.49	17.81	19.06	-1.15	20.91

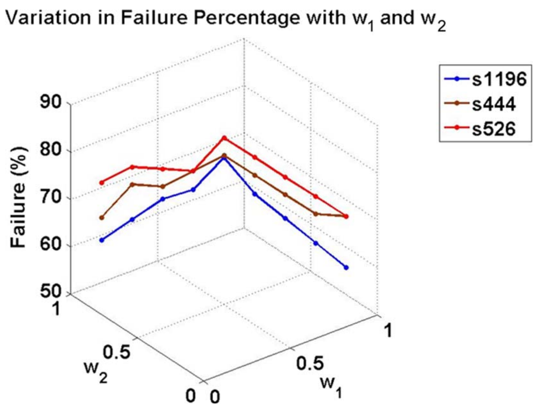


Fig. 9. Effect of weights w_1 and w_2 on verification failure percentage.

benchmarks were then subjected to formal verification using Synopsys *Formality*. Fig. 7 shows the observed percentage of verification nodes failing verification reported by *Formality* for the different benchmark circuits.

Effect of the iterative ranking algorithm: We also noted the improvement in our obfuscation methodology by adopting our multipass *iterative ranking* algorithm in place of a single-pass ranking algorithm. A general trend noticeable was that the percentage of nodes failing verification "saturated" with the increase in the number of modified nodes, irrespective of the ranking algorithm used. We observed mainly two effects of this iterative ranking algorithm. The first effect [Fig. 8(a)] was the increase in the percentage of reported failures (at equal area overheads) for the iterative ranking algorithm as compared to the single-pass algorithm. The second effect [Fig. 8(b)] was a decrease in the number of node modifications required to achieve the same maximum (saturated) failure percentage, so that we can identify the redundant node modifications and eliminate them, saving area overhead in the process. For example, from Fig. 7 for the s1488 benchmark, it is evident that we do not need to consider area overheads above 10%, because at higher area overheads, the same obfuscation level (in terms of the number of verification failures) is obtained.

Overheads incurred: Table III shows the area, delay, and power overheads of the resynthesized benchmark circuits, following the application of our obfuscation scheme, for the projected area overheads of 5%, 10%, 15%, and 20%, respectively. From the table, it is clear that the actual area overheads were smaller than the imposed constraints in all cases, while the timing overhead was negative, i.e., the timing constraint was met with positive slack in most cases. The power overheads in all cases were within acceptable limits. The design overhead is caused both by the addition of combinational (in the form of the *modification cells*) and few sequential elements to implement the inserted FSM. Although sequential memory elements do not scale as well as combinational ones, the percentage overhead is expected to remain unchanged for more advanced technology nodes because the combinational overhead forms the major fraction of the total overhead.

Optimal choice of parameters w_1 and w_2 in (7): We experimented with different combinations of the parameters w_1 and w_2 in (7), with $w_1 + w_2 = 1$. This was done to come up with a combination of w_1 and w_2 that will generate an optimal ranking of the nodes leading to maximum obfuscation. Fig. 9 shows the plot of the failure percentage for three benchmarks against different combinations of w_1 and w_2. From this plot, it is clear that a choice of $w_1 = w_2 = 0.5$ is optimal and leads to the maximum obfuscation. Such a trend was observed in all the benchmark circuits that we dealt with.

C. Results for the AES Encryption/Decryption Unit

Now, we present simulation results corresponding to the application of the *HARPOON* design methodology on the AES encryption/decryption core. We chose AES because it is widely deployed and is widely available from IP vendors as an RTL or gate-level IP. We took the open-source Verilog RTL implementations of 128-b AES encrypter and decrypter cores available in [18]. We modified the cores to integrate them in a single encrypter/decrypter core, the mode of operation being controlled by a *MODE_CONTROL* signal. The three main submodules of the design: the *Key Expand*, the *SBox*, and the *Inverse SBox*, were separately subjected to the design flow with an estimated area overhead of 5%.

The modules were designed with an integrated obfuscating FSM having four states and controlled by ten inputs each. The scheme shown in Fig. 5 was used, with an integrated controller

TABLE IV
OBFUSCATION EFFICIENCY AND DESIGN OVERHEADS (IN PERCENT) FOR THE AES ENCRYPTER/DECRYPTER MODULES (FOR 5% AREA CONSTRAINT)

AES Modules	Obfuscation Efficiency			Design Overhead		
	Nodes Modified (%)	Failing Verif. Points (%)	Failing Vectors	Area (%)	Delay (%)	Power (%)
Key Expand	0.95	78.3	4864	3.69	0.00	4.56
Sbox	0.95	100	236	3.62	2.42	5.31
Inverse Sbox	0.97	248	949	4.34	2.60	5.62

TABLE V
OVERALL DESIGN OVERHEADS (IN PERCENT) FOR THE AES CORE

Parameter	Original	Modified	Overhead
Area(μm^2)	2732060.0	2825636.6	3.43% increase
Power(mW)	565.0	586.6	3.82% increase
Start-up latency (cycles)	13	22	9 cycles increase
Op. Frequency(MHz)	230	220.1	4.30% decrease
Bit Rate(Gbps)	2.45	2.34	4.49% decrease

TABLE VI
DESIGN OVERHEADS (IN PERCENT) AT ISODELAY FOR ISCAS-89 CIRCUITS WITH AN EMBEDDED FSM UTILIZING UNREACHABLE STATES

Benchmark Circuit	Overhead (%)	
	Area	Power
s1196	18.44	15.88
s1238	16.31	15.83
s1423	6.11	9.65
s1488	12.81	6.22
s5378	14.45	23.84
s9234	6.53	9.04
Average	12.44	12.81

(which itself is a simple FSM) and a PUF module with five controlling inputs and two outputs. The controller is activated by a sequence of three inputs patterns applied at ten selected primary inputs. On activation, the controller starts reading the bits stored on an on-chip ROM in sequence. Simultaneously, it applies a 5-b pattern sequence to the on-chip PUF block. In response, the PUF block produces the 2-b sequences. In our HDL simulations, the PUF module was modeled as a ROM. These output bits from the PUF are XOR-ed with the outputs of the ROM and then applied to the different modules in parallel to activate them individually.

Design overhead: Table IV shows the performance and design overheads for the different modified modules, at an estimated 5% area overhead. Again, we find a very high rate of verification failures reported for the three modules, along with a large number of failing vectors. The actual area overhead was less than the estimated area overhead (5%) in all cases, and the power and timing overheads were within acceptable limits. The unmodified synthesized design had an operating clock frequency of 230 MHz and a data rate of 2.45 Gb/s. The modified design after resynthesis had an operating clock frequency of 220.1 MHz and a corresponding bit rate of 2.34 Gb/s. The controller FSM takes three clock cycles to get activated and then takes three clock cycles each to activate the *SBox* modules, the *Inverse SBox* modules, and the *Key Expand* module. Note that all the *SBox* modules are activated in parallel and so are all the *Inverse SBox* modules. Hence, the latency of the system for initialization is nine clock cycles, i.e., $\sim$41 ns. The start-up latency increases to 22 cycles from 13 cycles (the time taken to encrypt/decrypt one block of data) in the unmodified design. Note that this one-time increase in latency is incurred only at system start-up, which can be easily masked to the end-user. Table V shows the overall design overhead of the scheme. Again, we find all the overheads to be less than 5%, with the overall delay overhead slightly higher than the individual module overheads.

D. Mode Control Through Unreachable States

We can avoid the addition of extra SEs to realize the mode-control FSM by utilizing the *unreachable states* of a circuit. Complex sequential circuits would typically have a large number of unreachable states [20]. The state transition shown in Fig. 2 can be achieved if a few unreachable states are identified and the circuit is forced to go through these states during the initialization phase. By taking advantage of the unused state space of the original SEs, one can effectively reduce the design overhead and improve obfuscation level.

The required number of unreachable states would be equal to the number of patterns in the initialization sequence (e.g., three to four). In case a sufficient number of unreachable states are not identified, a combination of new states (due to extra SEs) and unreachable states can be used to realize the scheme. The unreachable states can be identified through simultaneous *sequential justification* of the SEs in the circuit. However, this is typically computationally intensive process. To avoid the difficulty of finding unreachable states for the entire set of SEs in the circuit, we note that random selection of a small subset of SEs from the circuit and their sequential justification is sufficient to identify unreachable states for the circuit. It also reduces the computational complexity of the problem drastically. Once the unreachable states are identified comprising of a set of SEs, a *parallel state machine* (PSM) can be formed using these SEs to realize the state transitions during the initialization process and then folding the PSM into the original one during resynthesis process. Additionally, the en signal is generated as an output of the PSM and alters the behavior of the circuit in the *obfuscated* mode through modification cells as described earlier. Upon successful initialization, all SEs hold values corresponding to a valid state, and the circuit enters *normal* mode.

To check the effectiveness of using unreachable states, we applied it to several ISCAS-89 circuits by choosing six SEs at random. The states unreachable by these six SEs were found through sequential justification by *Synopsys TetraMax*, and the RTL for a four-state PSM was generated from these unreachable states. The same number of modification cells as those used for the 10% area constraint were inserted. This RTL for the PSM and the modification cells was integrated with the gate-level netlist, and the circuit was resynthesized under delay constraint. Table VI shows the percentage area and power overheads at isodelay for the benchmark circuits. The fraction of nodes failing verification and the number of failing vectors were comparable with or better than the results presented in Table II. The overhead for some benchmarks such as *s1196* is high because the specific choice of six SEs and the state encoding using them resulted into large amount of extra logic in the synthesized netlist. By performing multiple iterations of random selection of SEs, it may be possible to find a different and/or smaller set of SEs which provides sufficient number of unreachable states. For example, we found a set of five and four SEs which led to an area overhead of 8.8% and 7.1%, respectively, both of which are considerably less than that obtained with additional SEs.

VI. Conclusion

We have presented a netlist-level hardware-obfuscation-based antipiracy design flow for SoCs. It involves active participation of the IP vendor, the IC designer, and the system designer and helps to preserve the rights of all of them. The scheme is based on the modification of the gate-level netlist of a presynthesized IP core, followed by resynthesis to obtain maximum functional and structural obfuscation at low design overhead. While providing obfuscation and authentication capabilities to the design at every stage of the design flow, the scheme does not affect the experience of an end-user. Simulation results with a set of ISCAS-89 benchmark circuits and the AES encryption/decryption core show that this scheme is capable of providing high levels of design obfuscation at nominal area overhead under delay constraint. The security of the design flow can be further enhanced by providing separate activation keys to each IP customer and using on-chip *PUF* circuits. Future work would target further reduction of design overhead and extension of the approach to higher level circuit descriptions, such as RTL.

References

[1] E. Castillo, U. Meyer-Baese, A. Garcia, L. Parilla, and A. Lloris, "IPP@HDL: Efficient intellectual property protection scheme for IP cores," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 5, pp. 578–590, May 2007.

[2] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Constraint-based watermarking techniques for design IP protection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 10, pp. 1236–1252, Oct. 2001.

[3] F. Koushanfar and G. Qu, "Hardware metering," in *Proc. ACM/IEEE Des. Autom. Conf.*, 2001, pp. 490–493.

[4] D. C. Musker, "Protecting and exploiting intellectual property in electronics," in *Proc. IBC Conf.*, 1998. [Online]. Available: http://www.jenkins.eu/articles/reverse-engineering.asp

[5] X. Zhuang, T. Z. Hsien-Hsin, S. Lee, and S. Pande, "Hardware assisted control flow obfuscation for embedded processors," in *Proc. Int. Conf. Compilers, Archit., Synth. Embedded Syst.*, 2004, pp. 292–302.

[6] *Thicket Family of Source Code Obfuscators*. [Online]. Available: http://www.semdesigns.com

[7] T. Batra, *Methodology for Protection and Licensing of HDL IP*. [Online]. Available: http://www.us.design-reuse.com/news/?id=12745\&print=yes

[8] R. Goering, *Synplicity Initiative Eases IP Evaluation for FPGAs*. [Online]. Available: http://www.scdsource.com/article.php?id=170

[9] *Xilinx IP Evaluation*. [Online]. Available: http://www.xilinx.com/ipcenter/ipevaluation/index.htm

[10] *ARM-The Architecture for the Digital World*. [Online]. Available: www.arm.com

[11] Y. Alkabani, F. Koushanfar, and M. Potkonjak, "Remote activation of ICs for piracy prevention and digital right management," in *Proc. Int. Conf. CAD*, 2007, pp. 674–677.

[12] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. Des. Autom. Test Eur.*, 2008, pp. 1069–1074.

[13] *Designware USB Solutions*. [Online]. Available: http://www.synopsys.com/products/designware/usb_solutions.html

[14] R. S. Chakraborty and S. Bhunia, "Hardware protection and authentication through netlist level obfuscation," in *Proc. Int. Conf. CAD*, 2008, pp. 674–677.

[15] F. Wang, "Formal verification of timed systems: A survey and perspective," *Proc. IEEE*, vol. 92, no. 8, pp. 1283–1305, Aug. 2004.

[16] W. A. Moore and P. A. Kayfes, "System and method to initialize registers with an EEPROM stored boot sequence," U.S. Patent 7 213 142, May 1, 2007.

[17] *The ISCAS-89 Benchmark Circuits*. [Online]. Available: http://www.pld.ttu.ee/~maksim/benchmarks/iscas89/verilog/

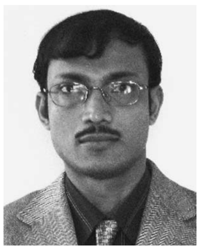
[18] *The 128-Bit Advanced Encryption Standard IP Core*. [Online]. Available: http://www.opencores.org/projects.cgi/web/aes_core

[19] *Recommended Practice for Encryption and [Use Rights] Management of Electronic Design Intellectual Property (IP)*. [Online]. Available: http://www.eda.org/twiki/bin/view.cgi/P1735/WebHome

[20] H. Yotsuyanagi and K. Kinoshita, "Undetectable fault removal of sequential circuits based on unreachable states," in *Proc. VLSI Test Symp.*, 1998, pp. 176–181.

Rajat Subhra Chakraborty (S'07) received the B.E. degree (with honors) in electronics and telecommunication engineering from Jadavpur University, Kolkata, India, in 2005. He is currently working toward the Ph.D. degree in computer engineering at Case Western Reserve University, Cleveland, OH.

He was a CAD Software Engineer with the National Semiconductor, Bangalore, India, where he has also held internship positions. He has also held internship positions at Advanced Micro Devices. His research interests include hardware security, including hardware IP protection, low-power very large scale integration, and nanocircuit design methodologies.



Swarup Bhunia (S'00–M'05–SM'09) received the B.E. degree (with honors) from Jadavpur University, Kolkata, India, in 1995, the M.Tech. degree from the Indian Institute of Technology, Kharagpur, India, in 1997, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 2005.

He has worked in the semiconductor industry on register-transfer-level synthesis, verification, and low-power design for about three years. He is currently an Assistant Professor of electrical engineering and computer science with Case Western Reserve University, Cleveland, OH. His research interest includes low-power and robust very large scale integration design, adaptive nanocomputing, and bioimplantable devices for neural engineering.

Dr. Bhunia was the recipient of the Best Paper Award in the Latin American Test Workshop in 2003, the Best Paper Award in the International Conference on Computer Design in 2004, the 2005 SRC Technical Excellence Award as a team member, and the Best Paper Nomination in the Asia and South Pacific Design Automation Conference in 2006. He has served in the technical program committee of the Design Automation and Test conference in Europe in 2006–2007, Test Technology Educational Program in 2006–2007, and International Symposium on Low Power Electronics and Design in 2007, and in the program committee of the International Online Test Symposium in 2005.