Structural Transformation for Best-Possible Obfuscation of Sequential Circuits

Li Li and Hai Zhou
Department of Electrical Engineering and Computer Science
Northwestern University

***Abstract*—Obfuscation is a technique that makes comprehending a design difficult and hides the secrets in the design. An obfuscation is called best-possible if the obfuscated design leaks no more information than any other design of the same function. In this paper, we prove that any best-possible obfuscation of a sequential circuit can be accomplished by a sequence of four operations: retiming, resynthesis, sweep, and conditional stuttering. Based on this fundamental result, we also develop a key-based obfuscation scheme to protect design Intellectual Properties (IPs) against piracy. The novel obfuscation method embeds a secret key in the power-up state of IC, which is only known by the IP rights owner. Without the key, the IC still functions but its efficiency will be much degraded. Unlike existing IC metering techniques, the secret key in our approach is implicit thus it can also be used as a hidden watermark. Potential attacks and the countermeasures are thoroughly examined, and experimental results demonstrate the effectiveness of the method.**

I. Introduction

The business model of semiconductor industry has changed significantly in last decade. With increasing complexity and cost of modern ICs, a design house has to seek the aid from various external agencies, such as EDA companies, IP vendors, library providers, and fabrication foundries. The active participation of external entities in the design and manufacturing flow has produced numerous hardware security issues. Among all the hardware security problems, piracy is likely to be the most ubiquitous and expensive one. Most leading edge design houses have outsourced their fabrication to the offshore foundries for the sake of lower labor and manufacturing cost. However, many offshore foundries are hard to be trusted since they are in country without consummate enforcement law for IP protection [1].

A variety of techniques has been proposed for fighting against hardware piracy. There are two main classes of approaches. One approach is hardware metering [2], which enables design houses to have post-fabrication control on the produced ICs. By metering, the designer can count the number of fabricated ICs, monitor their usage, and even remotely lock/unlock the ICs. Hardware watermarking [3], as another popular approach to IP protection, is inspired by the traditional digital watermarking technique. It inserts certain identity information into behavioral specification or sequential structure of the design. Watermarking is more passive compared with metering. But since watermarking has a unified signature for all ICs and does not involve any designer-manufacturer interaction, it will usually be less expensive.

Both hardware metering and watermarking techniques are intimately related to program/circuit obfuscation. Informally speaking, an obfuscator is a probabilistic compiler O that transforms a source program/circuit F into a new program/circuit $O(F)$ that has the same functionality as F but less intelligible in some sense. The technique of obfuscation is often used to protect the secrets in programs by making them harder to comprehend. However, circuit (hardware) obfuscation is radically different from program (software) obfuscation. Unlike that the functionality of a program can be kept secret, the functionality of commercial IC may be completely or partially known by parties other than the designer. The value of standard IC/IP is determined by their efficiency of implementation in terms of performance, power consumption, reliability, etc. Thus, instead of hiding the information within the original circuit, circuit obfuscation usually tries to hide extra secret information (e.g. watermarking) that is intentionally added to the circuit in order to prevent illegal use of the IC.

In this paper, we discuss two popular notions of obfuscation: black-box obfuscation [4] and best-possible obfuscation [5]. We believe that best-possible obfuscation is the appropriate definition in the context of hardware IP protection. Based on this definition, we show that any best-possible obfuscation of a sequential circuit can be accomplished by structural transformation composed of four types of operations: retiming, resynthesis, sweep, and conditional stuttering. Based on it, we propose a technique to obfuscate sequential circuits by structural transformation. The design will be embedded a secret key. It has two different work modes: normal mode and slow mode, depending on whether the initial state matches the key. In slow mode, the IC still functions but becomes much slower compared to normal mode. Since the normal mode contains an extremely small portion of all power-up states, any pirate who starts at a random state will most often end up using the degraded IC, without even suspecting the existence of the key. The hidden key can also be used as a proof of design authorship like watermarking while imposing extra penalty on piracy. Our contributions in this paper are listed as follows:

- We prove that retiming, resynthesis, sweep, and conditional stuttering are complete for any best-possible obfuscation of sequential circuits.
- We propose a key-based obfuscation scheme to hide the efficient implementation of the design and present the degraded design to the pirate.
- A number of potential attacks and the security of our method against them are discussed.
- Experimental results show the low overhead of timing, power, and area of our method.

The rest of the paper is organized as follows. The related works are discussed in Section II. The completeness proof of structural transformation for best-possible obfuscation is presented in Section III. A novel and practical obfuscation method is developed in Section IV. Attack resiliency is discussed in Section V. We show the experimental results in Section VI and conclude in Section VII.

II. Related Work

Program/circuit obfuscation is a fundamental problem in computer security. Barak et al. [4] initiated the theoretical study of obfuscation and demonstrated that generic "virtual black-box" program obfuscator does not exist. Later Lynn et al. [6] proved the first positive result about obfuscation, that the family of point and multi-point functions can be perfectly obfuscated under random oracle model. Goldwasser and Rothblum [5] argued that the black-box model be too strong for many real applications. They proposed a new notion of "best-possible" obfuscation under relaxed requirements and studied its

properties. Yet there is still lack of common agreement on the definition of obfuscation.

The concept of hardware metering is first introduced by Koushanfar and Qu in 2001 [2]. The idea was to assign an unique signature to the IC's functionality by making a small part of the design programmable. There followed some works that exploit manufacturing variability to generate unique random ID for each IC to achieve metering [7], [8], [9], [10]. These methods are all passive. Alkabani and Koushanfar [11], [12], [13] proposed the first active hardware metering scheme. The method utilized Physically Unclonable Function (PUF) [10] to generate the unique initial FF values (power-up state) for each IC. The power-up state will have very high probability to be in the non-functional part of an augmented FSM structure thus the IC will be locked. Only the designers who have knowledge about the augmented FSM structure would be able to send the key (transitions to legitimate reset state) to unlock the IC. According to a comprehensive survey about piracy avoidance [14], the methods based on embedding locks in the behavioral description of the design is also called internal active IC metering. In contrast, external active IC metering [15], [16], [17] usually embeds locks in the physical level of the design, which are further controlled by external cryptography function. The latter set of methods tend to have larger power and area overhead due to the complexity of cryptographic modules interfaced with the locks.

Oliveira first proposed to hide a secret watermark in a sequential circuit [3], [18]. The watermarking was performed by modifying the State Transition Graph (STG) to go through a chosen path of state transitions with certain set of inputs (secret keys). The insertion of watermark will not have any effect on the IC's functionality. The proof of authorship is ensured by the fact that the displayed input-transition behavior would be extremely rare in non-watermarked circuit. Later Koushanfar and Alkabani [19] proposed to add multiple watermarks to further enhance security, and they showed that hiding multiple watermarks in the STG is an instance of obfuscating a multi-point function with a general output. Yuan and Qu proposed the idea of hiding information in the unused transitions of FSM [20]. They developed a SAT-based algorithm to find the maximal set of redundant transitions for a given minimized FSM and took advantage of this redundancy to hide the information in the FSM without changing the given minimized FSM. Hardware watermarking looks similar as passive hardware metering but they have some critical differences. The watermarking signatures are uniform in all ICs of the same product in contrast to that metering will assign a specific signature for each IC. For this reason, watermarking can not track the number of fabricated copies from one mask.

III. Structural Transformation for Best-Possible Obfuscation

A. Best-Possible Obfuscation

The definition of obfuscation has been controversial for long since its theoretical study was initiated by Barak et al. [4]. Originally, obfuscation is defined in very strong requests that 1) the obfuscated circuit computes the same function as the original circuit with at most polynomial-time slow-down, 2) the obfuscated circuit should leak no information except for its black-box (input-output) functionality. Formally, black-box obfuscation requires that anything that can be efficiently computed from the obfuscated circuit, can also be computed efficiently from input-output access to the circuit. Barak et al. [4] showed that the general black-box obfuscator does not exist, indicating that the definition may be too strong. In fact, it is indeed too strong in the context of circuit obfuscation for IP protection. Since the obfuscated netlist is visible to the parties and its functionality may be publicly known, it is impossible to be treated as a black-box. Therefore, it is justified to use the notion of best-possible obfuscation proposed by Goldwasser and Rothblum [5]. The best-possible obfuscation relaxes the second requirement to that the obfuscated circuit leaks no more information than any circuit of the same functionality. Best-possible obfuscation guarantees that any information that is not hidden by the obfuscated circuit is also not hidden by any other circuit computing the same functionality. While this relaxed notion of obfuscation gives no absolute guarantee about what information is hidden in the obfuscated circuit, it does guarantee that the obfuscation is literally the best possible.

B. Functional Equivalence of Finite State Machines

Based on the definition, any obfuscated circuit must have the equivalent function as the original circuit. In this section, we will formally define *functional equivalence* between two circuits/FSMs.

Finite State Machine (FSM): FSM specifies how the system changes its states and produces outputs responding to inputs.

Definition 1: A FSM is quintuple $(Q, I, O, \lambda, \delta)$ where Q is a finite set referred to as the states, I and O are finite sets referred to as the set of inputs and outputs respectively, $\delta : Q \times I \rightarrow Q$ is the next-state function and $\lambda : Q \times I \rightarrow O$ is the output function.

Functional Equivalence: If we view a circuit as a black-box system, then its visible behavior can be described as its possible sequences of inputs and outputs. A circuit may exhibit an externally visible behavior like a sequence

$$\langle\langle E_0 = (I_0, O_0), E_1 = (I_1, O_1), E_2 = (I_2, O_2), ...\rangle\rangle$$

Note that in our specification every step in the sequence corresponds to a clock cycle in the sequential circuit. Traditionally, the equivalence of two FSMs [21] requires that their visible behavior should be precisely the same in every single clock cycle. In this paper, we will define this strict form of equivalence as *cycle-accurate equivalence* to avoid ambiguity.

Definition 2: Two FSMs C and C' are cycle-accurate-equivalent if any sequence of external behavior $\langle\langle E_0, E_1, E_2, ...\rangle\rangle$ that is allowed by C will be also allowed by C'.

Nevertheless, the relation of two FSMs computing the same function may not be restricted to cycle-accurate equivalence. If there exists internal states for the circuit, we can also have the complete behavior

$$\langle\langle (E_0, S_0), (E_1, S_1), (E_2, S_2), ...\rangle\rangle$$

Where S is the internal state (register values). In practice, sometimes only the internal state changes for example

$$\langle\langle (E_0, S_0), (E_1, S_1), (E_1, S_1'), (E_1, S_1''), (E_2, S_2), ...\rangle\rangle$$

Since the internal states are invisible to the users, the sequence of external behavior $\langle\langle E_0, E_1, E_1, E_1, E_2, ...\rangle\rangle$ and $\langle\langle E_0, E_1, E_2, ...\rangle\rangle$ compute the same function. Accordingly, we define the equivalence of two behavior sequences and derive the definition for equivalence of circuit behavior.

Definition 3: Two sequence of external behavior $\langle\langle E_0, E_1, E_2, ...\rangle\rangle$ and $\langle\langle E_0^*, E_1^*, E_2^*, ...\rangle\rangle$ are stuttering-equivalent if one can be obtained from the other by repeating states or deleting repeated states (by adding or removing finite amount of stuttering).

Definition 4: Two FSMs C and C' are functional-equivalent if for any sequence of external behavior $\langle\langle E_0, E_1, E_2, ...\rangle\rangle$ that is allowed by C, there exists a stuttering-equivalent sequence of external behavior $\langle\langle E_0^*, E_1^*, E_2^*, ...\rangle\rangle$ that is allowed by C'.

C. Structural Transformation

Previous approaches to circuit obfuscation usually operate on behavioral level of the design and require substantial modification on the STG of the design. The cost is potentially high since the STG will usually have exponential size in terms of the netlist. In this paper, we will focus on operating on structural level of the design. Our approach has lower cost since we do not need complete information of STG. We introduce four structural operations applied on sequential circuits: retiming, resynthesis, sweep, and conditional stuttering.

Retiming [22], [23] moves the registers in a sequential circuits while preserving its logic functionality. Two elementary operations can be applied: deleting a register from each input of a combinational node while adding a register to all outputs, and conversely adding a register to each input of a combinational node and deleting a register from all outputs.

Resynthesis restructures the netlist within the register boundaries without changing its logic functionality. Retiming becomes more powerful when combined with resynthesis due to the flexibility of changing register boundaries.

Sweep adds or removes registers affecting no output. Since synthesis normally simplifies the circuit structure, sweep is usually met as an operation removing redundant registers and logic.

Conditional stuttering is a newly defined structural operation in this paper. It adds control logic to the circuit to stutter the registers, i.e. copy the register values in the current cycle to the next cycle, under a given logic condition. The simplest implementation is to add a mux before the input of registers with two selections, the current register value and the next register value. Accordingly, inverse conditional stuttering can be defined as removing certain control logic.

D. Completeness Proof

Now we will show how to achieve any best-possible obfuscation of a sequential circuit by structural transformation. In [21], Zhou has demonstrated how to do cycle-accurate-equivalent transformation between two sequential circuits.

Lemma 1: If two circuits are cycle-accurate-equivalent, then one of them can be transformed to the other by a sequence of sweep (inverse), resynthesis, retiming, resynthesis, and sweep, given that the second resynthesis operation is allowed to use one-cycle reachability.

We will further extend the result in [21] and show that we can accomplish any functional-equivalent transformation by adding conditional stuttering.

Lemma 2: If two circuits C_1 and C_2 are functional-equivalent, then one can transform C_1 into a new circuit C_1' by conditional stuttering, and transform C_2 into a new circuit C_2' by conditional stuttering, such that C_1' and C_2' are cycle-accurate-equivalent.

With Lemma 1 and Lemma 2, it is not hard to draw the following conclusion.

Theorem 1: Retiming, resynthesis, sweep, and conditional stuttering are complete for structural transformation between any functional-equivalent circuits.

Corollary 1: Any best-possible obfuscation of a sequential circuit can be accomplished by a sequence of retiming, resynthesis, sweep, and conditional stuttering.

E. A Realistic Example

To better illustrate structural transformation for functional equivalence, we will use two small circuits that compute the Greatest Common Divisor (GCD) of two natural number as an example. The two original circuits (black lines) as shown in Figure 1 have the same functionality but different netlists due to different resource allocation. Circuit GCD_A is only assigned one subtracter while GCD_B is assigned two subtracters. Both have two registers. We omit the output part of both circuits, which outputs the value in either register when two register values are equal. The data flow description of two circuits are as shown in Figure 2.

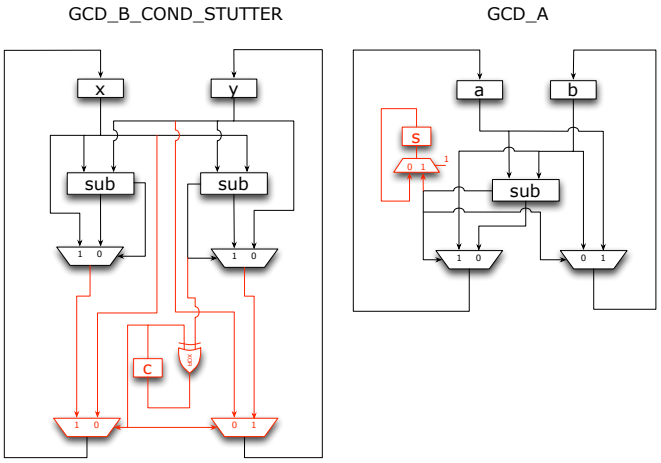

Fig. 1. Example of behavior-equivalent GCD circuits.

```
GCD_B(x,y)
while x!=y
  if x>y
    x:=x-y;
    y:=y;
  else
    x:=x;
    y:=y-x;
  endif
endwhile
return y;
```

```
GCD_A(a,b)
while a!=b
  if a>b
    a:=a-b;
    b:=b;
  else
    a:=b;
    b:=a;
  endif
endwhile
return b;
```

Fig. 2. Data flows of GCD circuits with different resources.

Our first observation is that GCD_A will use more cycles than GCD_B for the same computation because it needs extra cycles to swap the values in two registers when the subtraction result is negative. Thus GCD_A and GCD_B are functional-equivalent but not cycle-accurate-equivalent. However, if we apply conditional stuttering on GCD_B (red lines), the two resulted circuits will be cycle-accurate-equivalent. We can further apply sweep on GCD_A (red lines) such that we can find an onto state re-encoding between the two circuits.

$$F = \begin{cases} a = (c == 1)?y : x \\ b = (c == 1)?x : y \\ s = c \end{cases} \Leftrightarrow F^{-1} = \begin{cases} x = (s == 1)?b : a \\ y = (s == 1)?a : b \\ c = s \end{cases}$$

The same-length state re-encoding can be accomplished by retiming and resynthesis. Therefore, we can transform GCD_B to GCD_A by a sequence of conditional stuttering, resynthesis, retiming, resynthesis, and (inverse) sweep.

IV. Key-based Obfuscation Methodology

In last section, we showed the existence of structural transformation for any best-possible obfuscation of a sequential circuit. However, not any transformation will have realistic impact in terms of hardware IP protection. In this section, we will present a practical obfuscation scheme which is accomplished by a sequence of conditional stuttering, resynthesis, retiming, and resynthesis. The obfuscation hides a secret key to ensure that any pirate will only be able to use a degraded

design.

Among all structural operations, conditional stuttering makes the most difference for practice because it will significantly impact the effective performance (throughput) of the design. As we have mentioned, the functionality of a design, partially or as a whole, may not be secret in the context of hardware IP protection. The efficiency of implementation, i.e. performance, power, and reliability, then becomes the critical factor for evaluating the commercial value of the design. Based on this observation, we propose a key-based obfuscation scheme that makes the IC mimic another functional-equivalent IC whose throughput is much worse when the correct key is not provided. The behavioral similarity between the obfuscated IC with and without correct key is referred as bi-simulation. Our technique is mainly focus on those real-time applications that are very sensitive to throughput. We can see that there is a design trade-off for the performance difference in bi-simulation. On one hand, we hope the performance difference will be large enough so that the penalty for piracy will be substantial. On the other hand, the performance difference cannot be too apparent that the pirate would easily figure out the design is abnormal. The trade-off should be carefully considered in reality according to the features of specific applications.

A. Fundamental Framework

The IC can work in two different modes: normal mode and slow mode, depending on whether the circuit is initialized in the given key state. It can be guaranteed that there is only very slim possibility that an IC will work in normal mode if initialized randomly. Ideally, the existence of the secret key and different work modes should be only known to the IP owner. During initial power-up, the FFs will be initialized to a fixed state stored in non-volatile memory. When the design is sent for fabrication, the fab will only be informed of an initial state that is in slow mode. Thus if a malicious fab tries to illegally copy the IC, they will always end up using the bad design. After the IP owner receive the fabricated chips from the fab, the power-up state will be reset to the key state before the chips are sold.

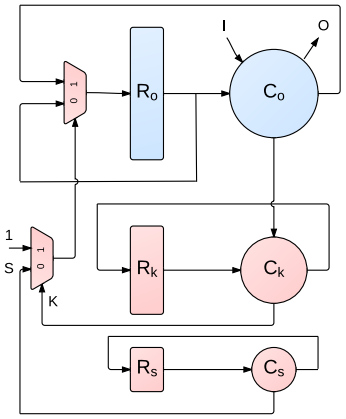

Fig. 3. Insertion of conditional stuttering circuit.

The first step of structural transformation of the design is an operation of conditional stuttering as shown in Figure 3. The blue components R_o and C_o are the registers and combinational logic of the original design, respectively. The red components are the stuttering control logic, which is composed of the key indicator (R_k, C_k) and the stuttering indicator (R_s, C_s). The key indicator will output an one-bit signal K, which will be 1 when the key is correct (the FF values of R_k is equal to the given key) and 0 otherwise. The stuttering indicator outputs a one-bit signal S. The final stuttering control signal is $K + \overline{K}S$. Thus the circuit will stutter if and only if $K = S = 0$, otherwise the circuit will work as normal. Note that the security strength of obfuscation is directly determined by the number of FFs in R_k. Assume there are k FFs in R_k and there only exists one set of FF values that will render $K = 1$, then a random power-up state will have a possibility of $1/2^k$ to fall in the normal mode.

B. Stuttering Control Logic

If the key indicator is utterly independent of the original design, then the only connection between the original design and the stuttering control logic will be an one-bit control signal. In this way the transformation may be more vulnerable to malicious attacks such as circuit partition. In order to enhance the security, we introduce some logic in C_o as the input of the key indicator to mix it up with the original design. In addition, we design the STG of the key indicator in such a way that the R_k will stay at the same state if and only if it is initialized to the correct key, otherwise it will be trapped in those non-key states (black hole states), as shown in Figure 4.

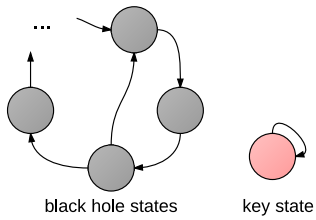

Fig. 4. STG construction for key indicator.

The stuttering indicator is used to control the average "frequency" of stuttering. Straightforwardly, the stuttering indicator can be designed as a simple counter that $R_s = (R_s + 1)\%2^s$, where s is the number of bits in R_s. Accordingly, the rule of stuttering can be set as $S = (R_s\%t == 0)?1 : 0$, where $1 - 1/t$ is the frequency of stuttering. In other words, the normal mode will be approximately t times faster than the slow mode to finish the same computation. The stuttering indicator can also be implemented in a non-deterministic way. For example, we can produce a random signal S to be 1 with $1/16$ possibility by using four random-bit gates with equal possibility to output 0 or 1. If we assume that the random-bit gates are hard to distinguish from normal gates, the non-deterministic implementation may be more favorable.

C. Secure Retiming and Resynthesis

The obfuscation prototype in Figure 3 has two practical flaws. First, the addition of stuttering control logic will result in extra delay, area, and power overhead. Since the control logic for all original FFs are changed, the total overhead can not be neglected. On the other hand, a smart adversary can possibly remove the stuttering control logic by logic dependency analysis. By carefully re-encoding the states, we can manipulate the dependency between the input and output of flip-flops thus make it harder to extract useful information from the netlist.[1] It is well known that any re-encoding of a sequential circuit can be done by a sequence of retiming and resynthesis as shown in Figure 5. The identity function at the register outputs is resynthesized to $F * F^{-1}$, where F is the one-to-one mapping from states of C to the target states of circuit $C^{'}$. Then retiming moves the registers forward over F. The last step resynthesizes $F^{-1} * C * F$

[1]See Section V for detailed discussion.

into $C^{'}$. Note that retiming and resynthesis will also help to reduce the overhead caused by adding stuttering control logic. Different re-encoding functions may be evaluated and the one resulting in the least overhead will be chosen as the final re-encoding function. The valid initial state (final key state) can be calculated from re-encoding function $F(R_o, R_k, R_s)$, where R_o is the valid reset state of original design, R_k is the selected key, and R_s is the initial counter value.

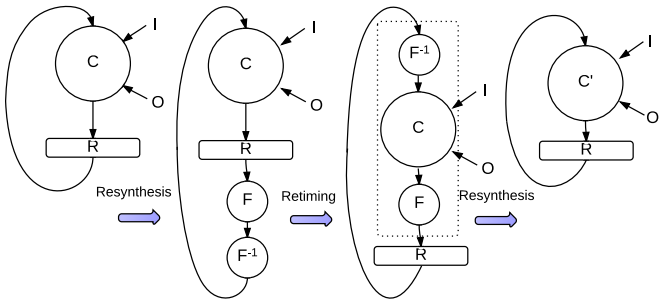

Fig. 5. Re-encoding of sequential circuits by retiming and resynthesis.

We choose linear transformation as our re-encoding function. An elementary linear transformation transforms the set of variables $X = \{x_1, ..., x_i, ..., x_j, ..., x_n\}$ into the set of variables $X = \{x_1, ..., x_i, ..., x_i \oplus x_j, ..., x_n\}$. An arbitrary linear transformation can be obtained by a series of elementary linear transformations, each one of them implementable by two xor gates, one gate in the transcoder before the registers (F) and one gate in the transcoder after the registers (F^{-1}).

V. Attack Resiliency

In this section, we enumerate possible attacks on our obfuscation scheme and discuss how the proposed method is secured against them.

- *Brute force attack*. The adversary attempts at randomly generating the power-up state until the throughput of tested IC is obviously better. Under the construction of our obfuscation, probability of finding the correct power-up state is extremely low. When the key indicator contains k FFs, the possibility of successful random guessing will only be $1/2^k$. Therefore, this class of attacks would not be able to break the security.
- *Stuttering control logic removal*. If the adversary is aware of the existence of the stuttering (though this cannot be observed through FF values), it may attempt to remove the stuttering control logic to regain a normal design. For the obfuscation prototype in Figure 3, the adversary can systematically analyze the logic dependency between all FFs and attempt to find out the added flip-flops for stuttering control. Let us denote the input and output of flip-flop i as D_i and Q_i, respectively. Then each D_i can be expressed as the combinational logic of a set of FF outputs $\mathcal{Q}_i$ and primary inputs. It is not hard to see that the majority of $\mathcal{Q}_i$ (original FFs) share a common subset of the outputs Q of stuttering control FFs. This characteristics can be exploited to find out all the FFs that belong to stuttering control logic, i.e., the set of stuttering control FFs can be obtained by the intersection of the inputs FFs $\mathcal{Q}_i$ for all FFs. However, re-encoding the states by linear transformation will be an effective countermeasure for the described attack. Linear transformation will help to decouple and rebind the logic dependencies. Here we give an example of two FFs A and B that share a common control signal S that

$$Q_A = D_A S + X_{A,B}\overline{S}$$
$$Q_B = D_B S + Y_{A,B}\overline{S}$$

Where X and Y are the next logic of Q_A and Q_B. After linear transformation that $A^{'} = A \oplus B$, $B^{'} = B$, we have

$$Q^{'}_A = (D^{'}_A \oplus D^{'}_B)Y_{A^{'},B^{'}} + D^{'}_B X_{A^{'},B^{'}}$$
$$Q^{'}_B = D^{'}_B S + Y_{A^{'},B^{'}}\overline{S}$$

It can be seen that FF A does not rely on control signal S any more. Though the actual combinational logic in our problem will be much more complicated, the decoupling effect of linear transformation is similar. Thus the attack by analyzing logic will be ineffective after linear re-encoding.
- *Inverse structural transformation*. The adversary may attempt to inversely transform the obfuscated IC into the original IC via structural transformation. However, without comprehensive knowledge about the design, the adversary can only randomly guess the re-encoding function and test if the function is possibly correct by stuttering control logic removal attack. In reality, this approach will be too expensive and time consuming for the purpose of piracy.

VI. Experimental Results

In this section, we report the overhead in terms of area, power, and timing of the synthesized circuits from the ISCAS89 benchmark suite. We first generate the original BLIF netlist of the benchmark circuits by ABC synthesis tool [24], which will be used as the baseline for obfuscated circuits. Then we will generate the BLIF netlist of the stuttering control logic. Finally the original circuit and stuttering control logic will be merged and obfuscated by resynthesis and retiming. All benchmark circuits are mapped to a standard cell library. In the experiments, we use 8 bits for the stuttering indicator and 24 bits for the key indicator.

Table I demonstrates comprehensive performance overhead evaluations on the ISCAS benchmark suite. The first column denotes the benchmark circuit name. The next three columns (Columns 2-4) show the original design statistics: the number of primary inputs, the number of primary outputs, and the number of FFs. Columns 5-7 demonstrate the design maximum delay in the following order: the original synthesized delay, the added delay post obfuscation, and the percentage of increase. The original designs power postsynthesis, the added power post obfuscation, and the ratio between the two are reported in Columns 8-10. The postsynthesis area of the original design, and the added area post obfuscation and the ratio between are shown in the last three columns, respectively.

We first analyze the impact of obfuscation on the circuit timing. From Figure 3, we can see that the critical path may be affected by newly added control signal, and the ratio of the added critical path delay overhead compared to the original delay seems to be independent of the circuit size. However, this overhead can be alleviated by the followed retiming and resynthesis optimization. Therefore, the actual overhead in the critical path delay introduced by our obfuscation is rather low, especially for large designs that have much flexiblity for retiming and resynthesis to leverage. For the tested cases with small or modest design size, on averge the delay overhead is 3.35 %.

The area and power overhead is very related in our method. In addition, they are not indepedent of the design size in the worst case since the control signal for all original FFs are changed. The area

TABLE I
EXPERIMENT RESULTS

Cases	Stats			Delay (ns)			Power (μW)			Area ($literal$)		
	PI	PO	FF	Ori	Incr	%	Ori	Incr	%	Ori	Incr	%
s382	3	6	21	0.463	0.032	6.9	161.1	118.3	73.5	255	195	76.3
s400	3	6	21	0.493	0.016	3.2	167.2	106.0	63.3	264	174	65.9
s526	3	6	21	0.434	0.042	9.6	190.4	136.9	72.2	338	225	66.7
s838	34	1	32	1.227	0.019	1.6	341.0	149.1	43.8	531	253	47.6
s953	16	23	29	0.911	0.046	5.0	391.8	156.6	40.0	595	263	44.2
s5378	35	49	179	0.736	0.021	2.8	1411.0	256.3	18.2	2248	512	22.8
s9234	36	39	211	1.755	0.034	1.9	2157.1	267.0	12.4	3492	543	15.5
s13207	62	152	638	1.86	0.028	1.5	4110.2	501.0	12.2	6339	1114	17.6
s15850	77	150	534	2.78	0.031	1.1	4565.7	590.9	13.0	7104	1316	18.5
s35932	35	320	1728	1.18	0.042	3.6	17787	1271.3	7.2	24934	2998	12.0
s38417	28	106	1636	1.46	0.021	1.5	11731	1242.7	10.6	18417	2923	15.8
s38584	38	304	1426	1.90	0.029	1.5	12458	929.3	7.5	20920	2179	10.4
average	-	-	-	-	-	3.35	-	-	31.2	-	-	34.4

and power overhead can be reduced by retiming and resythnesis, but not as much as in terms of delay. The overhead for area and power in our testcases are 31.2 % and 34.4 % on average. It can be seen that the overhead of our obfuscation scheme decreases as the size of the original design increases. Since our testcases are typically much smaller than current industrial designs, it can be estimate that the overhead for area and power will not exceed 10 % for realistic designs.

VII. CONCLUSION

This paper proved that any best-possible obfuscation of a sequential circuit can be accomplished by a sequence of retiming, resynthesis, sweep, and conditional stuttering. Furthermore, a novel obfuscation method for sequential circuits was presented to hide the efficient implementation of a design. Only when the IC is powered up in a secret key initial state that the IC will work normally otherwise it will become much slower. We showed how to construct the obfuscated circuits using the structural operations and improve the security of obufscation. We also discussed the possible attacks and the countermeasures. The efficiency of the method was demonstrated by evaluations on ISCAS89 benchmarks.

ACKNOWLEDGEMENT

This work is partially supported by NSF under CCF-1115550 and by NSFC under 61228401.

REFERENCES

[1] S. Trimberger, "Trusted design in FPGAs," in *Proceedings of the 44th annual Design Automation Conference*, 2007, pp. 5–8.

[2] F. Koushanfar and G. Qu, "Hardware metering," in *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 490–493.

[3] A. L. Oliveira, "Robust techniques for watermarking sequential circuit designs," in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, 1999, pp. 837–842.

[4] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang, "On the (im)possibility of obfuscating programs," in *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, 2001, pp. 1–18.

[5] S. Goldwasser and G. N. Rothblum, "On best-possible obfuscation," in *Proceedings of the 4th conference on Theory of cryptography*, 2007, pp. 194–213.

[6] B. Lynn, M. Prabhakaran, and A. Sahai, "Positive results and techniques for obfuscation," in *In EUROCRYPT 04*, 2004.

[7] K. Lofstrom, W. Daasch, and D. Taylor, "IC identification circuit using device mismatch," in *IEEE International Solid-State Circuits Conference*, 2000, pp. 372 –373.

[8] Y. Su, J. Holleman, and B. Otis, "A 1.6pj/bit 96% stable chip-ID generating circuit using process variations," in *IEEE International Solid-State Circuits Conference*, 2007, pp. 406 –611.

[9] F. Koushanfar, G. Qu, and M. Potkonjak, "Intellectual property metering," in *Inform. Hiding*, 2001, pp. 81–95.

[10] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proceedings of the 44th annual Design Automation Conference*, 2007, pp. 9–14.

[11] Y. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, 2007, pp. 20:1–20:16.

[12] F. Koushanfar, "Provably secure active IC metering techniques for piracy avoidance and digital rights management," *Information Forensics and Security, IEEE Transactions on*, vol. 7, no. 1, pp. 51 –63, feb. 2012.

[13] Y. Alkabani, F. Koushanfar, and M. Potkonjak, "Remote activation of ICs for piracy prevention and digital right management," in *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, 2007, pp. 674–677.

[14] F. Koushanfar, "Integrated circuits metering for piracy protection and digital rights management: an overview," in *Great lakes symposium on VLSI*, ser. GLSVLSI '11, 2011, pp. 449–454.

[15] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: ending piracy of integrated circuits," in *Proceedings of the conference on Design, automation and test in Europe*, 2008, pp. 1069–1074.

[16] ——, "Protecting bus-based hardware IP by secret sharing," in *Proceedings of the 45th annual Design Automation Conference*, 2008, pp. 846–851.

[17] J. Huang and J. Lach, "IC activation and user authentication for security-sensitive systems," in *Proceedings of the 2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 76–80.

[18] A. Oliveira, "Techniques for the creation of digital watermarks in sequential circuit designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, pp. 1101 –1117, sep 2001.

[19] F. Koushanfar and Y. Alkabani, "Provably secure obfuscation of diverse watermarks for sequential circuits," in *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2010, pp. 42–47.

[20] L. Yuan and G. Qu, "Information hiding in finite state machine," in *Proceedings of the 6th international conference on Information Hiding*, 2004, pp. 340–354.

[21] H. Zhou, "Retiming and resynthesis with sweep are complete for sequential transformation," in *Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design*, 2009, pp. 192–197.

[22] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.

[23] Y. Lu and H. Zhou, "Retiming for soft error minimization under error-latching window constraints," in *Design Automation and Test in Europe Conference*, 2013.

[24] R. Brayton and A. Mishchenko, "ABC: an academic industrial-strength verification tool," in *Proceedings of the 22nd international conference on Computer Aided Verification*, 2010, pp. 24–40.