Chapter 9
Structural Transformation-Based Obfuscation

Hai Zhou

9.1 Introduction

A variety of techniques have been proposed for fighting against hardware piracy. There are two main classes of approaches. One approach is hardware metering [15], 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 [21], 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. A program is usually obfuscated to hide its function, but the functionality of a commercial IC must be known to different parties other than the designer. The value of a hardware 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.

H. Zhou (✉)
Northwestern University, 2145 Sheridan Rd, Evanston, IL, USA
e-mail: haizhou@northwestern.edu

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_9

In this chapter, we discuss two popular notions of obfuscation: black-box obfuscation [3] and best-possible obfuscation [9]. Black-box obfuscation is stronger but has been proved impossible on general families containing point functions [3], while the best-possible obfuscation is weaker and possible to obtain [9]. Defined as disclosing only functionality, the best-possible obfuscation is more realistic in the context of hardware IP protection. Based on its 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. We then develop a Key-Locked OBfuscation (KLOB) scheme for hardware IP protection. In KLOB, a circuit will first be inserted with a stuttering logic with conditions both on key checking and on the state of the circuit. The conditionally stuttered circuit will then be further obfuscated by a sequence of retiming, resynthesis, and sweep operations. In the presence of the correct key value, the obfuscated circuit will run in the same speed as the original circuit; without the key, it will run in much slower speed. An simple version of the KLOB has been implemented to measure its overhead, and the effectiveness of the approach is thoroughly discussed.

9.2 Related Approaches

Program/circuit obfuscation is a fundamental problem in computer security. Barak et al. [3] initiated the theoretical study of obfuscation and demonstrated that generic "virtual black-box" program obfuscator does not exist. Later Lynn et al. [20] 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 [9] 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 [15]. The idea was to assign a 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 [16, 18, 27, 28]. These methods are all passive. Alkabani and Koushanfar [1, 2, 12] proposed the first active hardware metering scheme. The method utilized physically unclonable function (PUF) [28] 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 [11], 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 [10, 25, 26] usually embeds locks in the physical level of the design, which are further controlled

by external cryptography function. The latter set of methods tends 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 [21, 22]. 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 [14] 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 [29]. 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, while metering will assign a specific signature for each IC. For this reason, watermarking cannot track the number of fabricated copies from one mask.

9.3 Structural Transformation for Best-Possible Obfuscation

9.3.1 Best-Possible Obfuscation

The definition of obfuscation had been intuitive but not vigorous before its theoretical study was initiated by Barak et al. [3]. Barak et al. defined obfuscation in very strong requests that 1) the obfuscated circuit computes the same function as the original circuit with at most a polynomial-time slow-down and 2) the obfuscated circuit should leak no more information than its "black-box" (input–output invocation) functionality. Formally, "black-box" obfuscation requires that anything that can be efficiently learned from the obfuscated circuit can also be learned efficiently from input–output access to the circuit. Barak et al. showed that the general "black-box" obfuscator does not exist. The proof comes from the intuition that even an obfuscated program provides a complete function description, while a "black-box" oracle access may not be able to help to learn the complete function. This is especially true for point functions defined as follows:

$$C_{\alpha,\beta}(x) = \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{otherwise.} \end{cases}$$

More specifically, we can define another function $D_{\alpha,\beta}$ whose input is a function C:

$$D_{\alpha,\beta}(C) = \begin{cases} 1 & \text{if } C(\alpha) = \beta \\ 0 & \text{otherwise.} \end{cases}$$

Having the obfuscations of the two functions, $C_{\alpha,\beta}$ and $D_{\alpha,\beta}$, the adversary will apply the second function on the first one. The result is always one. However, if we only have "black-box" access to these functions, the probability for any simulator to get one is negligibly small. Therefore, there is no "black-box" obfuscation for any family that includes point functions.

The results of Barak et al. indicate that the "black-box" requirement in the definition may be too strong. In fact, it is indeed too strong in the context of circuit obfuscation for IP protection. When an IP block is provided, its functionality must be known and agreed up on by all parties. If it is a soft block, an obfuscated netlist is also visible to the parties. Thus, an IP block cannot be treated simply as a black-box. Following the study by Barak et al., Goldwasser and Rothblum [9] proposed a new definition of *the best-possible obfuscation* with a relaxed requirement in place of the "black-box" requirement. Intuitively, a best-possible obfuscation only leaks as much information as any circuit of the same function. In other words, it only leaks the functionality of the original circuit. 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 if the functionality is known.

Goldwasser and Rothblum also proved that there exists a best-possible obfuscation for a family of circuits that does not have "black-box" obfuscation. It shows that the definition of the best-possible obfuscation is strictly weaker than that of the "black-box" obfuscation. The family is the Polynomial-sized Ordered Binary Decision Diagrams (POBDD) [6]. Bryant [6] has shown that each OBDD has a canonical representation which can be efficiently computed. The best-possible obfuscation of any POBDD P is its canonical representation, which can be computed efficiently from any POBDD P' of the same function as P. Now, consider the point functions $C_{\alpha,1}(x)$ encoded in POBDD. As shown by Barak et al. [3], there is no "black-box" obfuscation for them.

9.3.2 Functional Equivalence of Finite State Machines

Starting from this section, we are going to show that the best-possible obfuscation can be computed by a sequence of structural transformations on the sequential circuit. Here, structural transformation means operations only on circuit netlist, not on state transition graph. 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

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

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 [30] requires that their visible behavior should be precisely the same in every single clock cycle. In this chapter, 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 $\langle\langle E_0, E_1, E_2, \ldots\rangle\rangle$ 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

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

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

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

Since the internal states are invisible to the users, the sequence of external behavior $\langle\langle E_0, E_1, E_1, E_1, E_2, \ldots\rangle\rangle$ and $\langle\langle E_0, E_1, E_2, \ldots\rangle\rangle$ 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 $\langle\langle E_0, E_1, E_2, \ldots\rangle\rangle$ and $\langle\langle E_0^*, E_1^*, E_2^*, \ldots\rangle\rangle$ 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 $\langle\langle E_0, E_1, E_2, \ldots\rangle\rangle$ that is allowed by C, there exists a stuttering-equivalent sequence of external behavior $\langle\langle E_0^*, E_1^*, E_2^*, \ldots\rangle\rangle$ that is allowed by C'.

9.3.3 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 chapter, we will focus on operating on structural level of the design. Our approach has lower cost since we do not need to generate any STG. All the operations are done on the circuit netlist. We introduce four structural operations applied on sequential circuits: retiming, resynthesis, sweep, and conditional stuttering. An example of applying the first three of them to transform one circuit into another is shown in Fig. 9.1. The example of conditional stuttering will be shown later.

Retiming [17, 19] 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 every output, or conversely adding a register to each input of a combinational node and deleting a register from every output. As can be seen from Fig. 9.1, retiming will change the state transition function and the state encoding while keeping the input/output functionality.

Resynthesis restructures the netlist within the register boundaries without changing its logic functionality. As seen from Fig. 9.1, resynthesis will not change the state transition, but it can create new signals in the circuit. These new signals can become new states if we move registers on them by retiming. The first resynthesis in Fig. 9.1 created two new signals for the subsequent retiming to use. Retiming becomes more powerful when combined with resynthesis due to new signals generated. Resynthesis also becomes more powerful when combined with retiming due to new combinational blocks generated by register moves.

Sweep adds or removes registers not having effect on the output. In Fig. 9.1, the sweep operation removes one register with the XOR gate since they do not affect the output. The sweep operation is necessary to change cycle lengths in the state transition graph. In Fig. 9.1 example, it reduces the length of the cycle in STG by half. Since synthesis normally simplifies the circuit structure, sweep is usually used as an operation to remove redundant registers and logic.

Conditional stuttering adds control logic to the circuit to stutter the registers, i.e., copy the register values in the current cycle to the next cycle, if a given logic condition is true. Stuttering is necessary if we want to transform a circuit into another that is not cycle-accurate-equivalent. It is easy to see that an obfuscated circuit can hide more information if it is not required to be cycle-accurate-equivalent to the original one. The simplest implementation is to add a multiplexer to the input of each register to select between the current register value and the next register value.

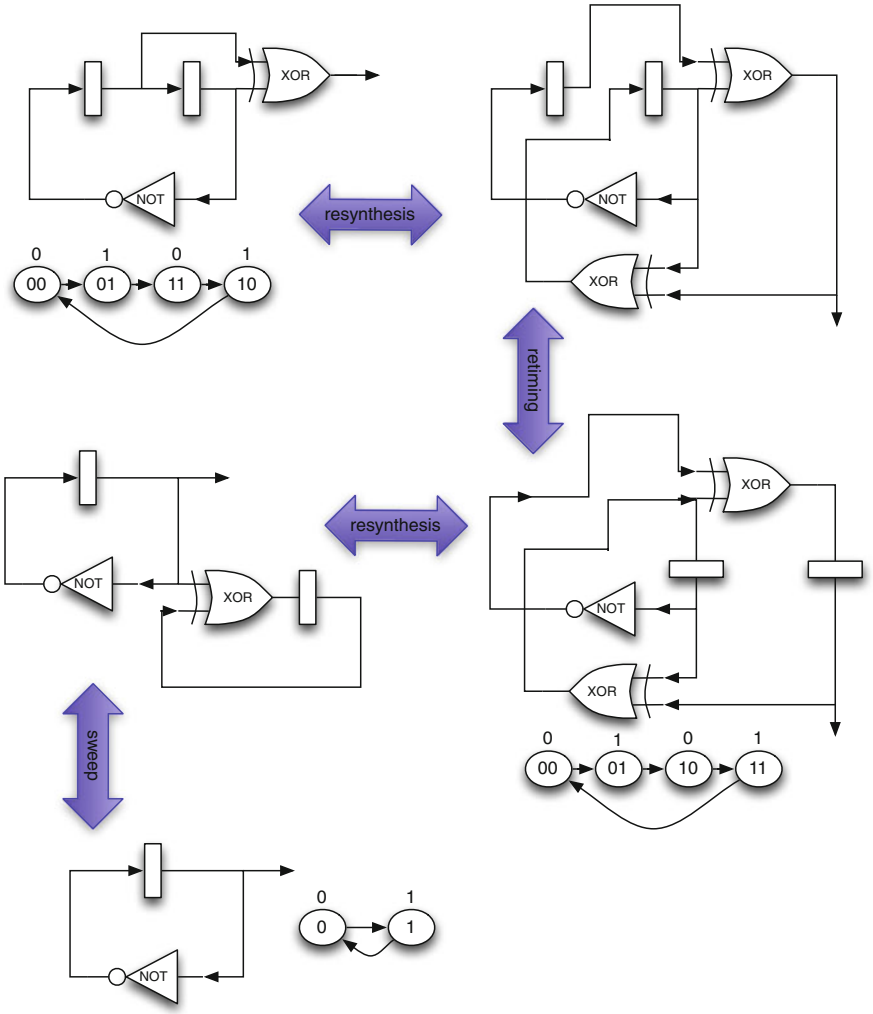

Fig. 9.1 Structural operations: retiming, resynthesis, and sweep

9.3.4 GCD Example for Conditional Stuttering

To better illustrate structural transformation for functional equivalence, we use two small circuits that compute the greatest common divisor (GCD) of two natural numbers as an example. They are different implementations of Euclid's algorithm. The two original circuits (dark lines) as shown in Fig. 9.2 have the same functionality but different netlists due to different resource allocation. Circuit *GCD_A* uses two subtracters, while *GCD_B* uses only one subtracter. Each circuit has registers for two integers. The basic iteration in Euclid's algorithm is to reduce the larger number

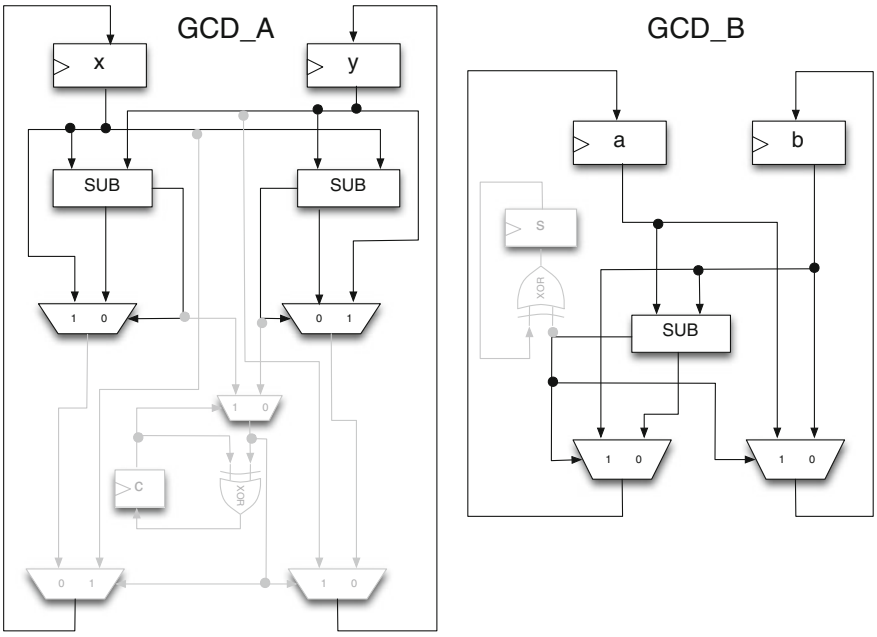

Fig. 9.2 The GCD example for conditional stuttering

to the difference of two numbers until they become the same. However, with only one subtracter, *GCD_B* may conduct a subtraction in wrong order, in which case, it needs to swap the two numbers. Because of it, *GCD_B* is slower than *GCD_A*. The circuits will output one number when they are the same. For simplicity, the outputs are not shown in the circuits.

Our first observation is that *GCD_B* will use more cycles than *GCD_A* for the same computation because it needs extra cycles to swap the two numbers if the subtraction result is negative. Thus *GCD_A* and *GCD_B* are functional-equivalent but not cycle-accurate-equivalent. In order to make them cycle-accurate-equivalent, *GCD_A* needs to be stuttering for one cycle when *GCD_B* is swapping the two numbers. Therefore, in order to know when *GCD_B* swaps, we need to keep track whether the number order in *GCD_A* is different from that in *GCD_B*. We introduce a register c for that purpose. Its value is 0 at the beginning and needs to be flipped when *GCD_B* swaps. The conditional stuttering in *GCD_A* is shown in gray lines in Fig. 9.2. With the conditional stuttering, the two circuits are cycle-accurate-equivalent. To make the mapping between the states of the two circuits explicit, we also introduce a history variable s in *GCD_B* by (inverse) sweep (shown in gray lines). It starts at 0 and flips when the number swaps. With these transformations, the mapping between the states of the two circuits is given as follows.

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

As an example, consider giving the input of 4 and 6 to the two GCD circuits. It means that $x = a = 4$ and $y = b = 6$ at the very beginning. The circuit *GCD_A* will generate a sequence of states (x, y) as (4, 6), (4, 2), (2, 2). The circuit *GCD_B* will generate a sequence of states (a, b) as (4, 6), (6, 4), (2, 4), (4, 2), (2, 2). As we can see that the two circuits are functional-equivalent since their final result is the same. However, they are not cycle-accurate-equivalent, since they take different numbers of cycles to reach the final states. But with the conditional stuttering in *GCD_A* and the history variable in *GCD_B*, the sequence of states (x, y, c) in *GCD_A* is (4, 6, 0), (4, 6, 1), (4, 2, 1), (4, 2, 0), (2, 2, 0), while the sequence of (a, b, s) in *GCD_B* is (4, 6, 0), (6, 4, 1), (2, 4, 1), (4, 2, 0), (2, 2, 0). It is easy to check that the corresponding states satisfy the mapping functions. The two cycle-accurate-equivalent circuits can be transformed from each other by a sequence of retiming and resynthesis. Therefore, we can transform *GCD_A* to *GCD_B* by a sequence of conditional stuttering, retiming, resynthesis, and sweep.

9.3.5 *Structural Transformation Sufficient for Best-Possible Obfuscation*

Now we will show that the best-possible obfuscation of a sequential circuit can be done by a sequence of structural transformations. As already demonstrated on a simple example in Fig. 9.1, we can transform any circuit into any other one that is cycle-accurate-equivalent to the original one. This is stated as the following lemma given by [30].

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.*

Similar to Fig. 9.2, it can be shown that conditional stuttering can transform a circuit into a circuit that is cycle-accurate-equivalent to any circuit that is functional-equivalent to the original one. Combined with the Lemma 1,

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

With Lemmas 1 and 2, we can show that any functional-equivalent transformation can be done if conditional stuttering is used in addition to retiming, resynthesis, and sweep.

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

The following corollary shows the existence of structural transformations for any best-possible obfuscation of a sequential circuit. It is based on the above theorem that structural transformations can derive any functional-equivalent circuit from a given circuit, and the definition that the best-possible obfuscation reveals at most information as any other equivalent program. Given the current state of art in behavioral synthesis and logic synthesis, we can safely state that giving a program (no matter what computational model it is on) is the same as giving an equivalent sequential circuit.

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

9.4 Key-Locked OBfuscation (KLOB)

The previous section shows the existence of structural transformation-based best-possible obfuscations. However, it does not provide a specific procedure, not even a guide, to do transformations for any best-possible obfuscation. The reason is that, even though Goldwasser and Rothblum [9] gave the definition of the best-possible obfuscation–one equivalent circuit, they did not show which one it is or not even which subset it belongs to. In this section, we will address this problem by developing a scheme called Key-Locked OBfuscation (KLOB).

9.4.1 KLOB Framework

We first argue that Key-Locked OBfuscation (KLOB) is the correct scheme for hardware IP protection. Based on the definition [9], the best-possible obfuscation of a circuit is one of equivalent circuits. Intuitively, in order to hide the original circuit, the obfuscation should be most different from the original one. But please note that it should not be *the* most different one if that helps to identify the original. Such a request helps to prevent the original to be understood or reverse-engineered. However, for hardware IP protection, that may not be sufficient: an adversary may not want to understand or modify the original, but to produce and use the circuit without permission. Therefore, an obfuscation that is very different from the original but with similar performance (speed, power consumption, etc.) is not very useful. However, if the obfuscation performs much worse than the original, then the legal users will suffer and complain. Therefore, any obfuscation for hardware IP protection should perform differently between an adversary and a legal user. And it is necessary to employ a secret key in the obfuscated circuit to differentiate the two modes, giving the Key-Locked OBfuscation (KLOB) scheme.

Behaviorally, the KLOB scheme works as follows. It uses a point function at the key value to select between two functional-equivalent circuits: the original one and its best-possible obfuscation with much worse performance. With the key, a legal user is served by the original circuit; without the key, an adversary is almost surely getting the much worse circuit. Of course, it is necessary to use obfuscation to mixed up the three parts of the circuit: the two versions of the circuit and the selection by the point function. Otherwise, an adversary may be able to extract the original circuit by analyzing the circuit.

Instead of starting with two equivalent circuits and then mixing them up, we will start with the original circuit and employ conditional stuttering to transform it. As shown in the previous section, stuttering based on circuit condition can mimic the behavior of any other equivalent circuit. If we only do this followed by a sequence of retiming, resynthesis, and sweep, what we can get is just a slower circuit of the best-possible obfuscation. In addition, KLOB does the stuttering also based on key checking. In other words, stuttering is happening in KLOB if and only if the key input is wrong *and* the circuit stuttering condition is true. Intuitively, the former encodes the selection by a point function at the key, while the latter encodes a slower circuit equivalent to the original one. The advantage of doing this is to make sure that the two circuits of the same function are tightly entangled together. After this conditional stuttering, a sequence of retiming, resynthesis, and sweep will be employed to make sure the three parts are inseparable.

The KLOB scheme after the conditional stuttering step is shown in Fig. 9.3. The components in dark color Ro and Co are the registers and combinational logic of the original circuit, respectively. The components in light color are the conditional stuttering logic, which includes the combinational circuit Cs to generate stuttering condition s based on the original state Ro and the extra registers Rs, and the key checking circuit Ck to generate the mismatching signal k. Only under the condition s AND k the original circuit is stuttering. Please note that Ck in a straight-forward design may not have the dashed connections in Fig. 9.3 and is only a combinational implementation of a point function that will only generate zero at a given key point.

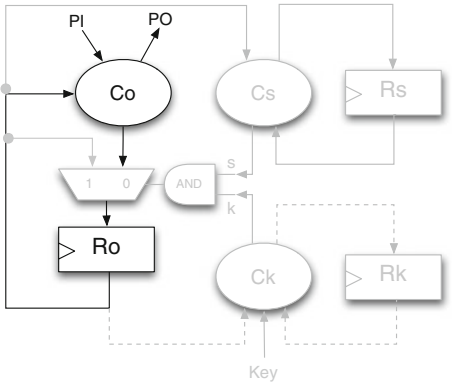

Fig. 9.3 Key-Locked OBfuscation (KLOB) scheme after conditional stuttering

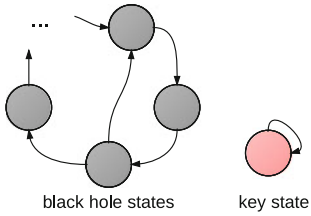

Fig. 9.4 Behavior of the key checking circuit

Such Ck is hard to be obfuscated by retiming and resynthesis, since there is only one bit and one-directional connection from Ck to the rest the whole circuit. In the next subsection, we will enhance the obfuscation of Ck by introducing extra registers Rk and the connection from Ro to Ck. With these modifications, it is easy to see that all the registers in Ro, Rs, and Rk can be retimed through or into Co, Cs, and Ck. It will greatly increase the security of the obfuscation by the followed retiming and resynthesis operations.

9.4.2 Stuttering Control Logic

This section will elaborate on the design of the stuttering control logic, including both Cs with Rs and Ck with Rk. As already mentioned, it is better to introduce extra registers Rk to Ck and to connect Ro to Ck to facilitate the obfuscation by later retiming and resynthesis operations. The idea is to make Rk to stay at the same state if and only if the correct key value is presented at the correct cycle, otherwise it will be trapped in the mismatch states (black hole states), as shown in Fig. 9.4. The transitions among the black hole states are dependent on Ro, making Rk depending on both Ro and Rk.

A simple design for stuttering signal s generation may make Cs with Rs as a counter such that $Rs' = (Rs + 1)\%2^w$ and set $s = (Rs\%t == 0)?0 : 1$, where $1 - 1/t$ is the frequency of stuttering. In other words, the slow circuit is approximately t times slower than the original circuit. However, such a design will make Cs independent of Ro, which will hurt the obfuscation of the whole circuit. From the structural point of view, it may limit the capability to retime registers on Cs; from the behavioral point of view, a very regular stuttering on the original circuit may only transform it into a very specific equivalent circuit, from which the original circuit may be easily derived.

As can be seen from the GCD example in Fig. 9.2, the right stuttering condition should be decided by the target circuit. The ideal approach is to follow the procedure shown in Fig. 9.2, that is, first come up with an equivalent circuit with much worse performance, then figure out a mapping between the states of the two circuits, and finally design the stuttering control logic to make the mapping one-to-one. An easier approach could first find an approximate formular of the reachable states in the original circuit (either by simulation or static analysis), then reformat it into a disjunction

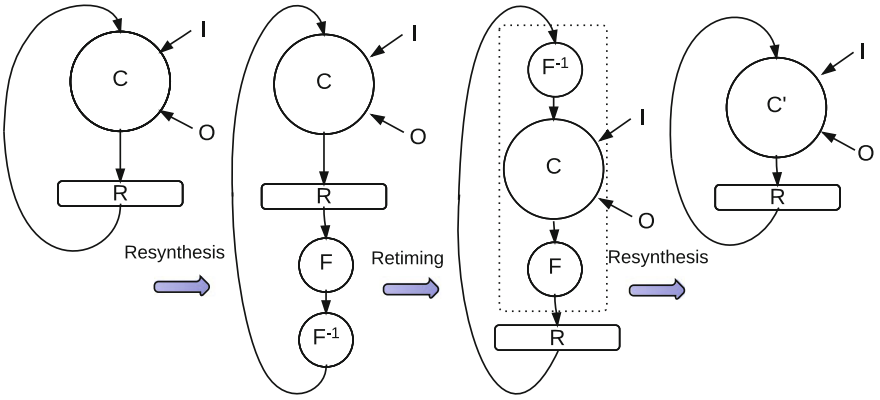

Fig. 9.5 Re-encoding of sequential circuits by retiming and resynthesis

of a few simple expressions, and finally give different numbers of stuttering cycles to different expressions.

Since the insertion of stuttering control logic results in extra delay, area, and power consumptions, its design has to consider not only obfuscation effect, but also the overhead. A good design must have a good trade-off between these effects.

9.4.3 Obfuscation by Retiming and Resynthesis

As shown in Sect. 9.3.5, conditional stuttering by itself is not sufficient for obfuscation. This can be easily seen on Fig. 9.3: by analyzing the circuit, an adversary can easily remove the stuttering control logic and get the original circuit! The conditionally stuttered circuit (shown in Fig. 9.3) has to be obfuscated by other structural transformations: retiming, resynthesis, and sweep. However, the transformation space by these operations is so huge, and it includes all cycle-accurate-equivalent circuits. We propose some basic ideas in this section. It should be note that any extra sequence of retiming and resynthesis operations can be applied on top of each other, and random operations of retiming and resynthesis can enhance the obfuscation.

It can be seen that one vulnerability of the conditional stuttered circuit in Fig. 9.3 is the relative independence of the three register groups Ro, Rs, and Rk. By carefully re-encoding the states, we can increase the dependency among them, 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 Fig. 9.5. 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

[1] See Sect. 9.5 for detailed discussion.

resynthesizes $F^{-1} * C * F$ into C'. Note that retiming and resynthesis may 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.

A linear transformation can be used as the re-encoding function. An elementary linear transformation transforms the set of variables $X = \{x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n\}$ into the set of variables $X = \{x_1, \ldots, x_i, \ldots, x_i \oplus x_j, \ldots, x_n\}$. An arbitrary linear transformation can be obtained by a sequence 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}).

9.4.4 Implementation Overhead

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 [5], 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 9.1 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 post-synthesis, the added power post-obfuscation, and the ratio between the two are reported in Columns 8–10. The post-synthesis area of the original design, 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 Fig. 9.3, 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 flexibility for retiming and resynthesis to leverage. For the tested cases with small or modest design size, on average the delay overhead is 3.35%.

Table 9.1 Experiment results

Circuits	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

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

9.5 Attack Resiliency

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

- *Brute force attack:* The adversary attempts at guessing the key until the throughput of tested IC is obviously better. It is well known that such an approach could be successful with very tiny probability.
- *Stuttering control logic identification:* Assume that the adversary knows that the circuit is obfuscated by KLOB, thus will try to identify the stuttering control logic. Running without the key, the circuit in Fig. 9.3 must have many stuttering steps in Ro, but not in Rs or Rk. This may be explored by the adversary to identify Ro. However, in KLOB, re-encoding and other retiming and resynthesis steps has been done on this circuit. Suppose A is a stuttering register in Ro and B is a changing register in Rs, a linear transformation $A' = A \oplus B$ will make A' changing, defeating the suggested attack.

It is already mentioned in Sect. 9.4.2 that it is better to make Rs and Rk dependent on Ro. Otherwise, since every register in Ro is dependent on Rs and Rk, a register dependence analysis may separate Ro from the others. Here again, even we did a bad job such that the dependence of Rs and Rk on Ro is weak, a linear transformation $B' = A \oplus B$ will make register B' dependent on register A. Therefore, a general linear transformation on all the registers in Ro, Rs, and Rk will also prevent register dependence analysis.

- *Inverse structural transformation:* The adversary may attempt to inversely transform the obfuscated IC into the original IC via structural transformation. However, without any knowledge on the obfuscation transformations, the adversary can only randomly guess the reverse re-encoding and transformations and test correctness by stuttering control logic identification. In reality, this attack is too expensive and time consuming for the purpose of piracy.
- *Key-based de-obfuscation:* Here, we consider an extreme case where the key has been somehow leaked and want to check how easy the adversary can get the original circuit. If no re-encoding or other retiming and resynthesis is done on the circuit in Fig. 9.3, applying the key can identify Rk since they are not changing. This can further help to identify k and s, thus to get the original circuit. However, with a thorough linear transformation on Ro, Rs, and Rk together, all the registers are mixed up, and

it is impossible to identify Ck. Therefore, we can safely say that, even when the key is leaked, its damage to a KLOB circuit is limited since the adversary can only use the original circuit but cannot get the design.

9.6 Conclusion

This chapter presents a circuit obfuscation technique called KLOB (Key-Locked OBfuscation) based on structural transformations. It first shows that any best-possible obfuscation of a sequential circuit can be accomplished by a sequence of retiming, resynthesis, sweep, and conditional stuttering. Then the KLOB is presented for hardware IP protection. Starting with an original circuit, KLOB first adds stuttering with conditions both on key checking and on the original circuit, and then obfuscates the conditionally stuttered circuit by a sequence of retiming, resynthesis, and sweep. With the correct key, the circuit will run in the original speed; otherwise, it will run much slower. The efficiency of the method was demonstrated by evaluations on ISCAS89 benchmarks. We also discussed the possible attacks and how KLOB is secure against them.

As we already mentioned, the benefit of structural transformations is to avoid the expensive STG manipulation. Therefore, the structural transformation-based obfuscation is more efficient than STG-based obfuscations. Logic obfuscation (also known as logic encryption) is a technique that uses a key and extra logic to modify the combinational design of a given circuit [4, 7, 8, 13, 23–25]. Since it only modifies the combinational logic, logic obfuscation can be viewed as a subset of the structural transformation-based obfuscation, where the allowed operations is only resynthesis.

For hardware IP protection, circuit performance is the key treasure to be protected. If the obfuscated circuit performs similarly as the original, an adversary will be happy to take it. If it performs much worse, then no user wants it. This means that the KLOB scheme is the right choice for circuit obfuscation: with the key, legal users get a circuit with the same performance as the original one; without the key, an adversary gets the best-possible obfuscation—an equivalent circuit with much worse performance. The theory in Sect. 9.3.5 ensures that, even when the key is known, an adversary will still not be able to get the original circuit. Our future work will study more sequences of structural operations to better obfuscate the conditionally stuttered circuit, especially the key checking circuit. We will leverage existing obfuscation techniques for point functions since key checking is essentially a point function.

References

1. Alkabani Y, Koushanfar F (2007) Active hardware metering for intellectual property protection and security. In: Proceedings of 16th USENIX security symposium on USENIX security symposium, pp 20:1–20:16

2. Alkabani Y, Koushanfar F, Potkonjak M (2007) Remote activation of ICs for piracy prevention and digital right management. In: Proceedings of the 2007 IEEE/ACM international conference on computer-aided design, pp 674–677
3. Barak B, Goldreich O, Impagliazzo R, Rudich S, Sahai A, Vadhan SP, Yang K (2001) (im)possibility of obfuscating programs. In: Proceedings of the 21st annual international cryptology conference on advances in cryptology, pp 1–18
4. Baumgarten A, Tyagi A, Zambreno J (2010) Preventing IC piracy using reconfigurable logic barriers. IEEE Design and Test 27:1
5. Brayton R, Mishchenko A (2010) ABC: an academic industrial-strength verification tool. In: Proceedings of the 22nd international conference on computer aided verification, pp 24–40
6. Bryant R (1986) Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput. 35:677–691
7. Chakraborty R, Bhunia S (2008) Hardware protection and authentication through netlist level obfuscation. In: IEEE/ACM international conference on computer-aided design
8. Dupuis S, Ba P-S, Natale GD, Flottes M-L, Rouzeyre B (2014) A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In: IEEE international on-line testing symposium
9. Goldwasser S, Rothblum GN (2007) On best-possible obfuscation. In: Proceedings of the 4th conference on theory of cryptography, pp 194–213
10. Huang J, Lach J (2008) IC activation and user authentication for security-sensitive systems. In: Proceedings of the 2008 IEEE international workshop on hardware-oriented security and trust, pp 76–80
11. Koushanfar F (2011) Integrated circuits metering for piracy protection and digital rights management: an overview. In: great lakes symposium on VLSI, GLSVLSI '11, pp 449–454
12. Koushanfar F (2012) Provably secure active IC metering techniques for piracy avoidance and digital rights management. Inf. forensics and secur., IEEE Trans. 7(1):51–63
13. Koushanfar F (2012) Provably secure active IC metering techniques for piracy avoidance and digital rights management. IEEE Trans. on Inform. Forensics and Secur. 7:1
14. Koushanfar F, Alkabani Y (2010) Provably secure obfuscation of diverse watermarks for sequential circuits. In: IEEE international symposium on hardware-oriented security and trust (HOST), pp 42–47
15. Koushanfar F, Qu G (2001) Hardware metering. In: Proceedings of the 38th annual design automation conference, pp 490–493
16. Koushanfar F, Qu G, Potkonjak M (2001) Intellectual property metering. Inform. Hiding. Springer, Heidelberg, pp 81–95
17. Leiserson CE, Saxe JB (1991) Retiming synchronous circuitry. Algorithmica 6(1):5–35
18. Lofstrom K, Daasch W, Taylor D (2000) IC identification circuit using device mismatch. In: IEEE international solid-state circuits conference, pp 372–373
19. Lu Y, Zhou H (2013) Retiming for soft error minimization under error-latching window constraints. In: Design automation and test in Europe conference
20. Lynn B, Prabhakaran M, Sahai A (2004) Positive results and techniques for obfuscation. In: In EUROCRYPT 04
21. Oliveira AL (1999) Robust techniques for watermarking sequential circuit designs. In: Proceedings of the 36th annual ACM/IEEE design automation conference, pp 837–842
22. Oliveira A (2001) Techniques for the creation of digital watermarks in sequential circuit designs. IEEE Trans. on Comput.-Aided Design of Integr. Circuits and Syst. 20(9):1101–1117
23. Rajendran J, Pino Y, Sinanoglu O, Karri R (2012) Security analysis of logic obfuscation. In: Design automation conference
24. Rajendran J, Zhang H, Zhang C, Rose GS, Pino Y, Sinanoglu O, Karri R (2015) Fault analysis-based logic encryption. IEEE Trans. on Comput. 64:2
25. Roy JA, Koushanfar F, Markov IL (2008) EPIC: ending piracy of integrated circuits. In Design, Automation and Test in Europe
26. Roy JA, Koushanfar F, Markov IL (2008) Protecting bus-based hardware IP by secret sharing. In: Proceedings of the 45th annual design automation conference, pp 846–851

27. Su Y, Holleman J, Otis B (2007) A 1.6pj/bit 96% stable chip-ID generating circuit using process variations. In: IEEE international solid-state circuits conference, pp 406–611
28. Suh GE, Devadas S (2007) Physical unclonable functions for device authentication and secret key generation. In: Proceedings of the 44th annual design automation conference, pp 9–14
29. Yuan L, Qu G (2004) Information hiding in finite state machine. In: Proceedings of the 6th international conference on information hiding, pp 340–354
30. Zhou H (2009) Retiming and resynthesis with sweep are complete for sequential transformation. In: Proceedings of 9th international conference on formal methods in computer-aided design, pp 192–197