Hardware Obfuscation (Drawn from "Hardware Protection thr Obfuscation)

Circuit modification techniques designed to **conceal** or **lock** the functionality and/or circuit structure

The modifications are designed to increase the difficulty for an adversary to *reverse engineer* the circuit and *clone* the IC/IP and/or *insert backdoors*

The methods discussed here are *active* in preventing reverse engineering/piracy from occurring

In contrast to watermarks, which are *passive*

Borrowing from the software world, an *obfuscator O* is a compiler that transforms a program *P* into *O(P)* such that

- *O(P)* computes the same function as *P*
- Anything computed in poly time from *O(P)* can be computed given 'oracle' access to *P* (*O(P)* can be used as a 'virtual black box')

Note that unlike encryption that obfuscates data, a general approach to obfuscating all programs does not exist

Threats

There are primary three threats that can be addressed by obfuscation techniques

- IP/IC Piracy

  Unprotected HDL, netlist, FPGA bitstreams and layouts can simply be copied into other designs

  Overbuilding and coping does not require reverse engineering, the entire IP/IC is simply copied and sold on the black market

- Reverse Engineering

  RE involves analyzing layouts and netlists to obtain higher level (HDL) descriptions of the original design

  This allows adversaries to steal clever design components and/or study vulnerabilities in the original design

- Trojan Insertion

  In the most effective cases, RE is first performed to identify places to insert information leakage channels and/or kill switches

Encryption

Standards have been developed to protect IP in some cases

For example, FPGAs allow IP blocks to be encrypted, and then used within designs

The IP owner holds the key and the CAD tool, e.g. Vivado, unencrypts the IP for use in design flows

Also, FPGA bitstream encryption is widely used to protect the entire design, and FPGA programming procedures enable on-chip decryption using embedded keys

However, in general, it is difficult to use encryption to protect IP, particularly in ASIC design flows

In nearly all steps of the design flow, unencrypted transparency is required by the synthesis and place&route tools

Therefore, alternative obfuscation methods and/or logic locking strategies are needed to prevent IP/IC piracy, overbuilding, RE and hardware Trojan insertions

RTL Level Obfuscation

Soft IP in the form of verilog/VHDL behavioral descriptions can be encrypted as described above, and decrypted by CAD tools

Note that although these techniques provide protection against direct copying from one design to another, they do not protect against overbuilding

Moreover, once the IP is decrypted and integrated into a design, it can remain unencrypted through the remaining steps of the design flow

Therefore, RE attacks can be performed in later design stages

RTL-level Locking techniques are proposed to deal with all of these threats, i.e., piracy, RE and Trojan insertion

RTL locking integrate the **key** into the design itself, by modifying the data flow and/or state transition diagram by adding dependencies on the correct key

During startup, the modified design traverses through a set of states based on the key to reach a valid starting state

RTL Level Obfuscation

White-box obfuscation can also be used, where the HDL itself is *scrambled* and *transformed* to make it difficult to read while maintaining functionality

Note that these techniques may increase the difficulty of RE and Trojan insertion, they do not protect against overbuilding

Only key-based methods are able to provide *hardware metering*

Gate Level Obfuscation

Gate level IP is usually referred to as **firm IP**, and is represented as a netlist with wires and standard logic cells

Logic Encryption is a technique which adds extra gates, e.g., XOR, XNOR and MUX to the netlist

At least one of the inputs of these extra gates are driven by **key** bits

The embedded key is stored in tamper-resistant NVM or are derived from PUFs

An incorrect key causes incorrect logic values to be generated during functional operation

Security is provided because only the IP owner knows the correct key

Therefore, such methods protect against piracy, RE and Trojans

Unfortunately, these methods are vulnerable to **SAT-based attacks** and ATPG-oriented key propagation attacks

Adversaries attempt to propagate keys to observe points, e.g., primary outputs

Gate Level Obfuscation

FSM-based Locking embeds an *authenticating FSM* that requires a specific sequence of input patterns, that represents the key

A modification kernel function is inserted to obfuscate the combinational components if the traversal is not successful

Re-synthesis is used mix-up the obfuscating components with the design components, making the RE process more difficult

Re-use of unreachable states in the FSM is proposed as a means of keeping the overhead low

However, the method still exhibits high penalty in speed, power, area

Gate Level Obfuscation

Protocol level techniques have been proposed in which the locking method utilizes a *key exchange* mechanism with a trusted 3rd party

The techniques are pitched to prevent IC overbuilding and IP piracy

Such methods are generally referred to as **hardware metering**

FSM-based locking can be used in combination with PUFs for generating a unique key per chip

In this case, the foundry transmits the key to the IP owner, the IP owner *computes* the proper unlocking sequence and transmits it back to the foundry

The PUF bits are used in combination with the unlocking sequence to enable a chip-specific unlocking scheme

These techniques introduce **black-hole** states into the FSM, which when entered, cannot be backed out of

These methods are costly in terms of overhead

Layout Level Obfuscation

Layout level IP is commonly referred to as **hard IP**, and is represented as a set of masks which define the geometries of wires and transistors

Split-Manufacturing is proposed in which the untrusted foundry fabricates front-end-of-line (FEOL) components

The wafers are then sent to the trusted design house to add the remaining metal wiring layers, i.e., the lower-tech back-end-of-line (BEOL) processes

The method is designed to hide *connectivity information* from the untrusted foundry

However, connectivity can be derived in some cases based on knowledge of the CAD tool behavior, i.e., connecting wires in proximity, etc.

These methods are costly to perform, and require close tolerances to be maintained between the untrusted and trusted foundries

Wire-lifting is proposed for 3D (TSV) technologies and interposers for 2.5D

Layout Level Obfuscation

Camouflaging has been proposed to prevent RE

Here, special camouflaged std cells are designed that make their function, NAND, NOR, XOR, etc. ambiguous

Dummy contacts and doping manipulation are common camouflaging techniques

Fundamental Assumptions:

- RE technology is 2-3 generations behind latest CMOS design technology
- A top-down image processing-based attack which makes dummy and true contacts indistinguishable

These techniques are expensive in terms of overhead, and are susceptable to SAT-based attacks

Layout Level Obfuscation

A *configurable cell* is used to disguises the functionality of selective XOR, NAND and NOR gates:

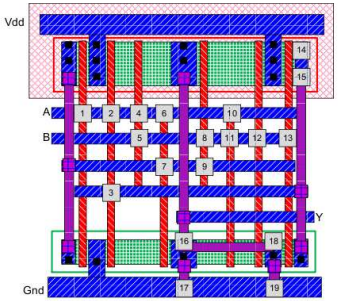

Figure 4: A generic camouflaged layout that can perform either as an XOR, NAND or NOR gate based on which contacts are true and dummy. The numbered boxes are the contacts.

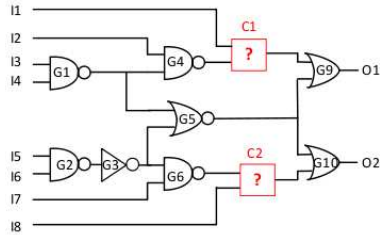

Figure 5: C1 and C2 are isolated camouflaged gates. An attacker can resolve the functionality C1 and C2 independent of each other.

ATPG can easily be used to distinguish cells unless *interference* exists between camouflaged cells, i.e., these cells are in series

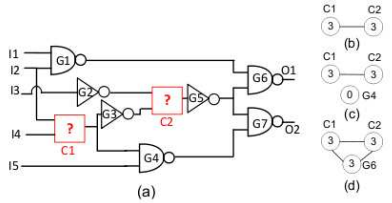

Figure 8: (a) A design camouflaged with 2 gates: C1 and C2, (b) Corresponding interference graph, (c) Interference graph with C1, C2, and G4 camouflaged, and (d) Interference graph with C1, C2, and G6 camouflaged. The number below the node label indicates the weight of a node.

Claim is that you need 3^m guessing attempts (exhaustive) to determine correct instantiations

Unfortunately, partitioning the circuit into sub-circuits breaks it quickly