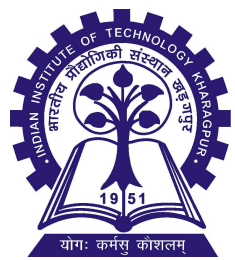
A Survey on Gate Level Information Flow Tracking



Computer Science and Engineering
Indian Institute of Technology, Kharagpur, India

Secure Embedded Architecture Laboratory (SEAL), 2014

Outline

1. Overview
2. Application of GLIFT in RTL Level Security Vulnerability Detection
3. Issues with GLIFT

1 Overview

2 Application of GLIFT in RTL Level Security Vulnerability Detection

3 Issues with GLIFT

Information Flow Control

- "Information Flow Control (IFC) is a security paradigm which classifies data objects into different security levels and monitors the propagation of data among security domains to prevent sensitive information from leakage or high-integrity data from being violated. IFC can be implemented either through static information flow analysis or dynamic information flow tracking (IFT) to enforce certain information flow security policies such as Bell LaPadula for confidencility and non-interference for integrity" [1].

Information Flow Control

- Till date, numerous works have been done on static information flow analysis at the programming language and compilar levels [2], standard OS primitives [3], [4] and at the ISA/microarchitecture levels [5], [6], [7].

Information Flow Control

- Till date, numerous works have been done on static information flow analysis at the programming language and compilar levels [2], standard OS primitives [3], [4] and at the ISA/microarchitecture levels [5], [6], [7].

- However, the above mentioned methods, although effective for detecting some harmful information flows and security vulnerabilities, are all at too high a level of abstraction to identify hardware-specific timing channels, which has been shown to cause secret key leakage in stateful components such as caches [8] and branch predictors [9].

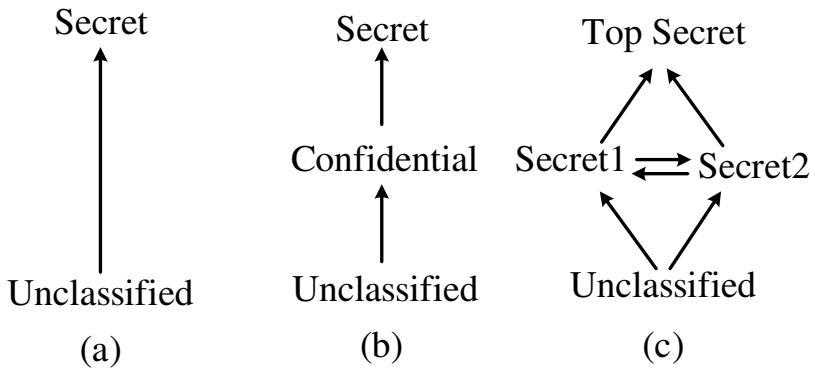
Information Flow Control

- Till date, numerous works have been done on static information flow analysis at the programming language and compilar levels [2], standard OS primitives [3], [4] and at the ISA/microarchitecture levels [5], [6], [7].
- However, the above mentioned methods, although effective for detecting some harmful information flows and security vulnerabilities, are all at too high a level of abstraction to identify hardware-specific timing channels, which has been shown to cause secret key leakage in stateful components such as caches [8] and branch predictors [9].
- Solution: Fine grained information flow tracking.

Gate Level Information Flow Tracking

- Gate Level Information Flow Tracking (GLIFT) has been proposed in [10].
- GLIFT enforces some information flow policy on the system and tracks all the direct and indirect flows of information throughout the system, using tagged data bits and an well formed tracking logic, till the granularity of the gate level.
- The GLIFT tracking logic has been shown to provably detect covert timing information flows in cache, bus systems, and processors [10, 11, 12, 13].
- It has also been shown to detect the violation of 'non-interference' policy in mix-trusted SoCs, where, the condition was to ensure that the trusted information flow should never be mixed with the untrusted one [14, 15].

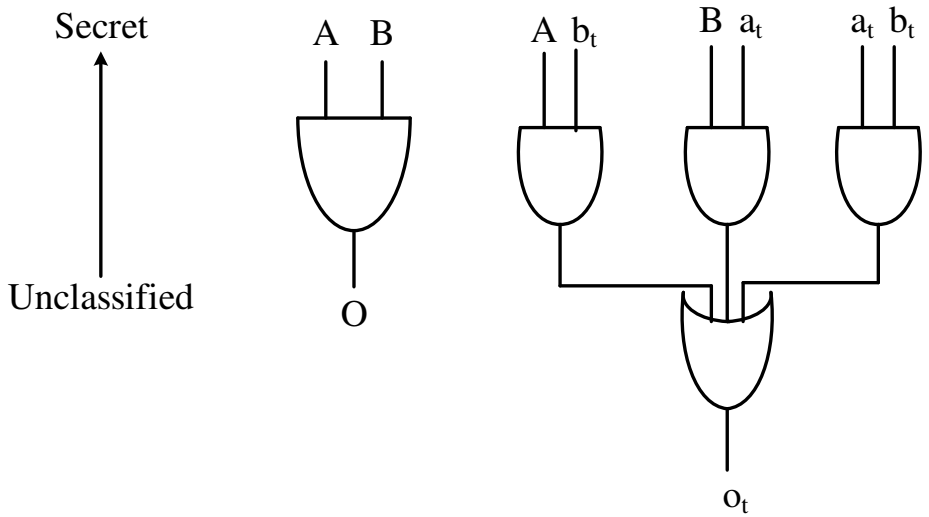
Gate Level Information Flow Tracking (cont.)



- In the GLIFT logic, each binary data bit is associated with a label to indicate its security level. The labels are decided according to some security lattice structure enforcing some security policy.
- Each security class in the security lattice is associated with one single label/tag.
- For example, in the case of the two level security lattice (Fig.(a)), each data bit will be associated with one bit tags/labels 0→ Unclassified; 1→ Secret.
- For other complex security lattices, there will be multi-bit encoding for the labels/tags.

GLIFT Tracking Logic

- GLIFT framework defines tracking logic for each gate given a specific security lattice.
- Let us consider a two-input and gate and a simple two-level security lattice for illustration. The two inputs A and B and the output O of the *AND* gate have associated tag/label bits a_t, b_t and o_t, respectively.
- Tag bit value 0 indicates that the associated data bit is *unclassified* and value 1 indicates that the associated data bit is *secret*.



- The corresponding GLIFT logic contains 4 gates.

Intution behind GLIFT Tracking Logic

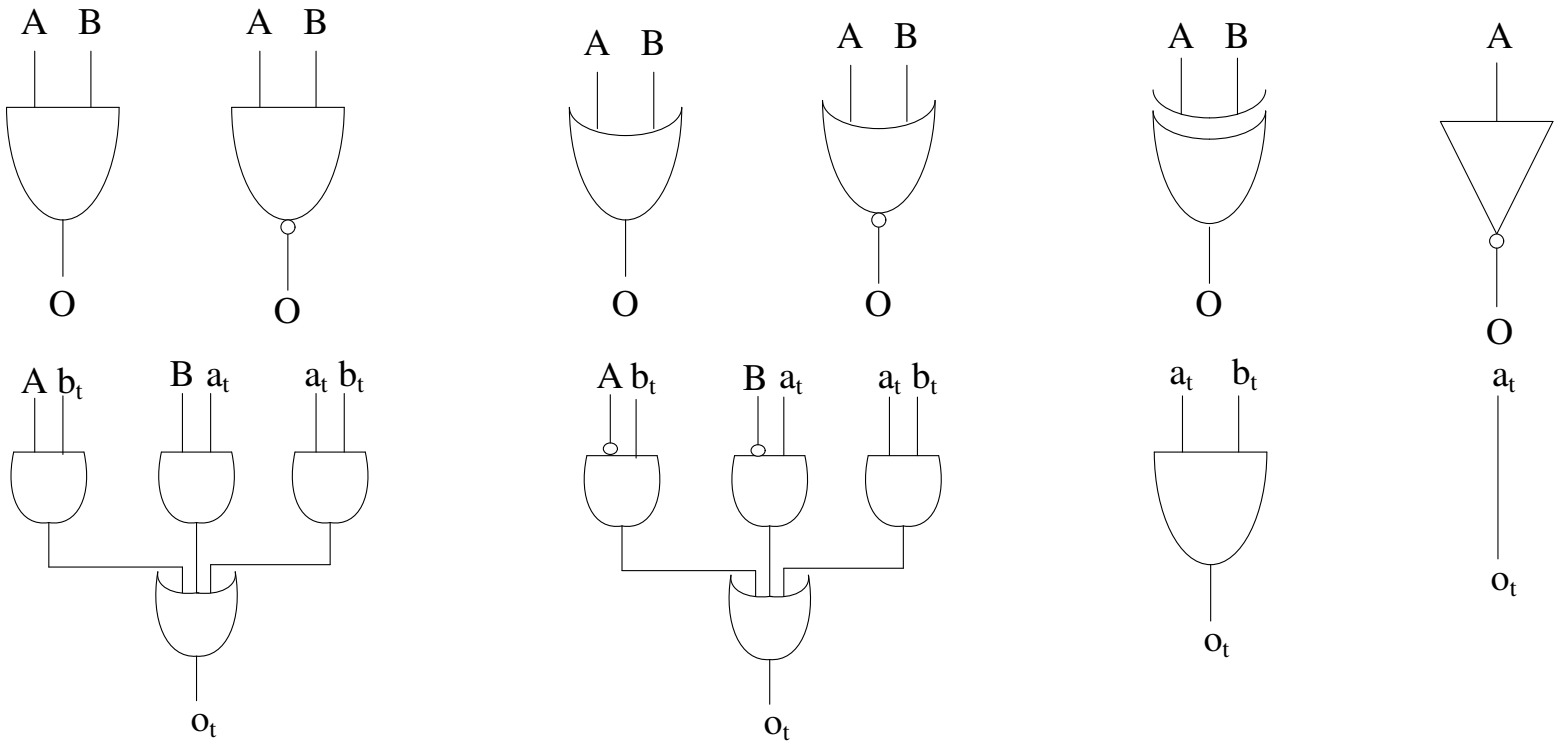
Table: GLIFT Logic AND operation with three Encoding States

A	a_t	B	b_t	O	o_t
0	0	0	1	0	0
0	0	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	1

The natural intuitions behind the tracking logic are:

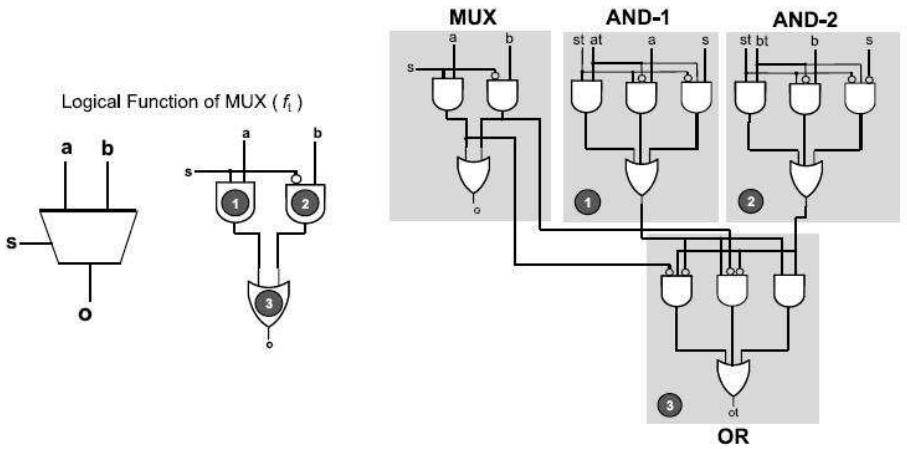
- The security label of the output is bounded to the most restrictive security class among the input security classes, whose data actually affects the output.
- If both the inputs have tag secret the output is always secret.
- If both the inputs have tag unclassified the output is always unclassified.
- If one of the input is secret and another is unclassified then the output tag depends on the data value of the unclassified input (see the table above).

GLIFT Tracking Logic for Basic Logic Gates



Composing Larger Functions with GLIFT

Composition based GLIFT Logic Generation for 2-input MUX



Generally two approaches exist:

- Composition Based: Simply instantiates GLIFT logic for each single gate of some complex logic.
- Suffers from some logical hazards in some cases which increases the probability of false alarm. However, it never misses some unwanted information flow.
- Binary Decision Diagram (BDD)Based: Creates the BDD of the gate level logic and then adds extra prime implicant terms to construct the logic. this approach removes the above mentioned false alarm problem.

1 Overview

2 Application of GLIFT in RTL Level Security Vulnerability Detection

3 Issues with GLIFT

GLIFT in RTL Level Security Vulnerability Detection

- GLIFT has been prominently used to ensure *non-interference* between secret and unclassified (or trusted and untrusted) data. However, enforcement of security lattices also exist.
- In practice, certain system states (e.g. cache misses) often indirectly leaks some useful information about the ongoing computation. GLIFT can effectively detect them.
- GLIFT can detect unwanted indirect information flow via processor states [10, 11, 12].
- Using GLIFT, it can be proved that if critical system states (e.g. program counter state, branch predictors, cache states etc.) are under the influence of some untrusted/unclassified data in some instant of computation, all the data in the system will be marked as untrusted.
- The above mentioned fact is utilized for cache-timing attacks on crypto primitives.
- GLIFT can detect any such security loophole during RTL and Gate level design.

GLIFT in RTL Level Security Vulnerability Detection (cont.)

- GLIFT has also been used for the integration of mix-trusted third-party IP cores [14, 15].
- The key idea is to avoid any unwanted information flow (ensuring non-interference) while integrating some trusted IP core with some other third-party IP cores which may contain security loopholes (like Hardware Trojans).
- Strict non-interference ensures that even indirect timing information flows are prohibited.

1 Overview

2 Application of GLIFT in RTL Level Security Vulnerability Detection

3 Issues with GLIFT

Theoretical Issues with GLIFT

- GLIFT works as a security verification tool.
- It can be shown that the resource requirement of the GLIFT logic increases exponentially with the increasing complexity of the security lattice as well as the increasing fan-in of the gates [16, 17].
- The problem of precise(i.e. optimized and false alarm free) GLIFT logic generation for larger circuits is NP-complete [16, 17].

Practical Issues with GLIFT

- For Static verification, where the extra tag propagation logic need not to be synthesized, GLIFT is well suited.
- However, in some cases, (e.g. general purpose processors where the control flow is not constant) Dynamic Security verification might be needed. In those cases, GLIFT may add significant amount of hardware overhead.
- Even with the most optimized implementation available till date [18], it adds 3x hardware overhead for each single gate.

Thank You