Gate-Level Netlist Reverse Engineering for Hardware Security: Control Logic Register Identification

Travis Meade*, Yier Jin*, Mark Tehranipoor†, and Shaojie Zhang*
*Department of Electrical Engineering and Computer Science, University of Central Florida
†Department of Electrical and Computer Engineering, University of Florida

***Abstract*—The heavy reliance on third-party resources, including third-party IP cores and fabrication foundries, has triggered the security concerns that design backdoors and/or hardware Trojans may be inserted into fabricated chips. While existing reverse engineering tools can help recover netlist from fabricated chips, there is a lack of efficient tools to further analyze the netlist for malicious logic detection and full functionality recovery. While it is relatively easy to identify the functional modules from the netlist using pattern matching methods, the main obstacle is to isolate control logic registers and reverse-engineering the control logic. Upon this request, we proposed a topology-based computational method for register categorization. Through this proposed algorithm, we can differentiate data registers from control logic registers such that the control logic can be separated from the datapath. Experimental results showed that the suggested method was capable of identifying control logic registers in circuits with various complexities ranging from the RS232 core to the 8051 microprocessor.**

I. Introduction

Globalization has drastically changed the entire integrated circuit (IC) supply chain. A modern computer system may include chips that have had some portion of their manufacturing (fabrication, assembly, etc.) performed in many different parts of the world, leaving it extremely difficult to trace the origin of each component [1]. Furthermore, due to the fact that IC fabrication heavily relies on oversea foundries and the heightened awareness of how chips are vulnerable to hardware Trojan insertion at said foundries, security concerns have significantly risen recently.

In order to ensure the trustworthiness of the fabricated chips, various hardware Trojan detection methods have been proposed [2]–[4]. The detection accuracy is the main obstacle of the previously developed non-destructive approach. Therefore, the chip-level reverse engineering becomes the last resort to solve the Trojan detection problem, despite its high cost. Thanks to the advanced imaging techniques, reverse engineering companies (e.g, Chipworks [5]) can easily derive the full netlist from a fabricated chip. Designers who have the full design information may compare the original design with the reverse engineered netlist to identify whether any additional logic has been inserted during the fabrication process. However, this method may not apply to the commercial-of-the-shelf (COTS) ICs and SoCs with third-party IPs because of the lack of golden models.

Restoring the trust in the suppliers can be achieved by assisting the users in determining and, potentially, verifying chip functionality. Besides the straightforward design comparison method, a more sophisticated design analysis tools is needed. The tool should be capable of using a recovered netlist as the input and then recover the chip's full functionality. Relying on these tools, users can verify the trustworthiness of purchased ICs. Various tools have been developed recently in this area [6]. These tools treat all internal registers the same way and try to identify the functional logic through pattern matching. As a result, the identified functional components mix the control logic and the datapath. This leaves the testers with the difficult task of fully recovering the design's functionality.

Given this situation, this paper presents a novel method for analyzing and classifying the registers of an arbitrary unlabeled netlist. Employing this method a user is capable of accurately identifying the control logic registers of a netlist, whether they are malicious or intended. It will be assumed that the chip design can be obfuscated, and techniques to sanitize the chip and diminish the effects of such chip design will be described. After this pre-processing, an algorithm for scoring registers will be presented that aids in grouping registers and, eventually, classifying the groups as logic registers and non-logic registers. The registers' scores will be generated in an unsupervised fashion, with no reference list of common module structures. This is done to help accurately analyze chips that use a set of unintentional logical structures.

The remainder of this paper is organized as follows: we will first cover current work related to state register identification and classification of netlist in Section II. There will be a brief overview of this paper's proposed tool (RELIC) followed by a detailed description in Section III. Section IV will present a method for determining the effectiveness of RELIC. A thorough analysis of three netlists will then be presented along with an analysis of the parameters and their affects on RELIC in Section IV. This paper will then wrap up with Section V which will give a brief summary of the paper and present ideas for different applications of RELIC.

II. Related Works

One of the main issues for reverse engineering digital circuits is determining state registers. State registers, or control logic, affect what a circuit will do with both the stored data on the chip and the data passed in through the inputs. Subtle manipulation of a circuit's state registers can be destructive and dangerous to users of these afflicted cores. Even worse, in order to lower the time to market (TTM), many ICs are fabricated by untrusted third parties these days. One problem with brute force checking each manufactured chip is the amount of time it takes. In a situation where each register is independent of each other and can either have '1' or '0' value, a chip with only 100 registers has over 10^{30} states.

Knowing the registers of the netlist's state logic is crucial to understanding the chip's functionality, malicious or not. Many tools for specification recovery require having prior knowledge of the state registers [7], [8]. Using these state registers the listed methods can reconstruct part of the chip's functionality. Methods have been proposed for state register recovery. For example, the authors in [7], [9] utilized a similar method for finding such registers. Both methods classify a register as a state register if its output could indirectly affect itself. This technique reduces to classifying a register as logic if it is in a connected component. Although simple, it provided a good and fast baseline for determining logic registers. It should be noted, this technique fails to identify any data registers, if all registers are in the same connected component.

Another strategy was to determine word-level registers. When groups of registers are identified as words certain registers that affect their output will normally be sought after state registers. An example of that was the subgraph isomorphism between subgraphs of the circuit and a component library used in [10]. If the subgraphs were identified, then their function would become known, which would lead to knowledge of the individual registers as well. This technique does not perform well, if the netlist was made with the intention of not having similar subgraphs to the libraries provided. Also obfuscating the graph by adding useless combinational gates between important gates can prevent subgraph isomorphism from properly matching subgraphs. WordRev [6] used the intersection of bounded fan-in subtrees of gates to find control logic. This method fails to identify words if the subtree is too small or the control logic is spoofed.

These methods show that there is a need for identifying a netlist's important logic registers. To fulfill said need, this paper aims to classify registers quickly using a scoring function generated by examining the logical and topological similarity between pairs of registers in the netlist. Other methods have been proposed that classify netlists, potentially infected with Trojans, based on a scoring metric [2]. This noteworthy paper also used structure checking to generate values for its metric. However, unlike RELIC, the proposed method in [2] compared sub-structures of the given netlists to known Trojan structures. This reliance on a set of known structures could potentially fail to identify novel Trojans. Our tool, RELIC, removes the dependency of a structure library and classifies registers as logic or data based on a self-structure analysis in an attempt to detect potential novel Trojan structures, while maintaining an accuracy similar to other methods on known Trojan structures.

III. RELIC

Reverse Engineering Logic Identification and Classification (RELIC), as previously mentioned, takes in an arbitrary netlist and produces a list of "Similarity Scores" for each register gate pairs. These scores allow for a classification of registers that are important (e.g., a Trojan register or an intended chip control logic register). This paper covers a method for generating Similarity Scores through the use of a mixture of Dynamic Programming techniques and advanced graph algorithms, thereby creating a type of pseudo graph isomorphism. However, rather than comparing against a secondary graph structure, the structures of the netlist's registers are compared against each other. This technique can also be adapted to comparing other netlists to an original one. The idea is based on an observation that register pairs from the same datapath

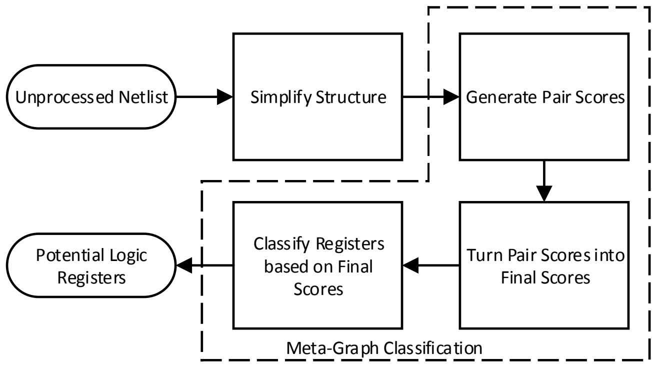

Unprocessed Netlist
Simplify Structure
Generate Pair Scores
Turn Pair Scores into Final Scores
Classify Registers based on Final Scores
Potential Logic Registers
Meta-Graph Classification

Fig. 1: Flow Diagram for RELIC

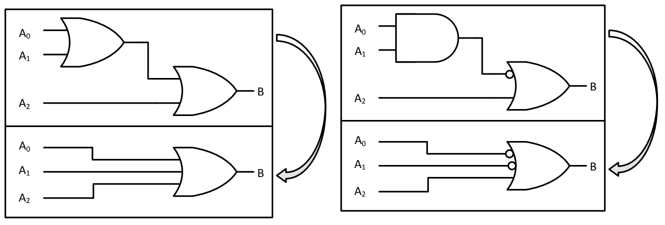
will have a similar deep logical structure. Two data registers with the same word rely on the same control logic registers to propagate their signals along the same datapath from fan-in to output. The corresponding data signals on the datapath most-likely come from a similar logical structure from the previous clock cycles. A similar logic structure will be present all the way to the input signals. By taking advantage of the similarity of the data logic registers' fan-in structure, RELIC was developed to accurately analyze a netlist that has a datapath controlled by potentially obfuscated signals. This method identifies chip logic registers by finding registers that are not exclusively part of any datapath. Lastly this technique can help find malicious logic inserted by others, since Trojan logic registers normally have logic different than other original register logic structure.

RELIC was developed to replace the graph isomorphism approaches with a faster heuristic by loosely comparing the topology of the fan-in logic. Due to its pseudo-isomorphism's fuzzy logic, RELIC can match registers of the same word with a higher accuracy than traditional word checking methods that require the logic to meet a very specific structure. Figure 1 shows the work flow diagram for RELIC. It can identify registers, or even words, that are similar, but if there is an obvious word that is improperly connected within the chip, RELIC might allow it to go undetected. This tool can be used in combination with functional testing (or another lower level tool of the sort) to verify its findings.

A. Preprocessing

Taking an arbitrary pair of logic vertexes (registers or logical gates) RELIC will generate values that represent how similar their fan-in logic structure is to each other. The most obvious thing to do is to check if the logic function used by the logic vertexes is equivalent. If this preliminary check fails, a score of near zero is given to the pair. However, this check is strict. For example, NOR and AND have similar output types. Also an XOR can be simulated by an OR and two AND gates. In these two cases registers can easily have the same logic but have varying raw structures. Thus RELIC uses a preprocessing step to reduce the structure complexity, e.g., all XOR gates will be reduced to AND-OR-INV logic.

When designing a netlist some gate level obfuscation might occur, either purposefully or accidentally. Figures 2a and 2b show simple examples of functional obfuscation. There are two main scenarios. The first scenario is when the input logic



(a) Similar gate types. (b) Differing gate types.

Fig. 2: Different types of gate level obfuscation.

vertex has a similar AND-OR logic to that of its parent logic vertex, and its output is not inversed (see Figure 2a). The second scenario is when the input logic vertex is inversed, and it has a different AND-OR (see Figure 2b), then the child logic vertex can be merged. In the first part of preprocessing, the fan-in is checked for potential inputs that can be combined. If so, all wires can then be merged, and this process is repeated until the logic vertexes cannot merge with any of their un-merged children. Obviously no registers have their logic merged with a logical gate, because we assume that registers update only on one edge of the clock cycle.

After the above step, a color is given to INPUT, AND, OR, Register AND, and Register OR logic vertexes. Additionally when checking two logic vertexes one might also want to check if the structure of one is similar to the inverse of the other. This can be simulated by swapping AND color vertexes with OR color vertexes and vice versa, in the inversed logic fan-in subgraph. This color/logic swap can then be used as a preliminary verification that the inversed logic vertexes have the potential to be similar.

B. Generating Similarity Scores Through Topological Analysis

RELIC generates similarity scores for an arbitrary pair of logic vertexes. Each score will fall in the range from 0 to 1, where scores of 1 will denote identical fan-in structure, 0 will be no common structure. These scores will be obtained by determining the similarity of all pairs of inputs between the logic vertexes in question. A connection will be added to a bipartite graph, if the score was above a predetermined threshold. A matching algorithm is then used to find the maximum disjoint children pairs that are similar between the original logic vertexes in the constructed bipartite graph. Figure 3 shows an example of the bipartite matching of the two wires.

After finding the maximum matching of the bipartite graph, the similarity score (number of matches) for the given wire pair is normalized by the maximum number of inputs between the two logic vertexes ($max(n, m)$). Re-computing similarity scores can hurt run-time performance, especially if a logic vertex has a large fan-out set. To prevent re-computations, a dynamic programming technique of memoization is used.

One question that arises is "How does this procedure handle infinite recursion?" Based on the current scoring functions, a pair of logic vertexes that each contain a fan-in path affected by their respective outputs might not halt on RELIC. To prevent this infinite loop from occurring a user defined depth,

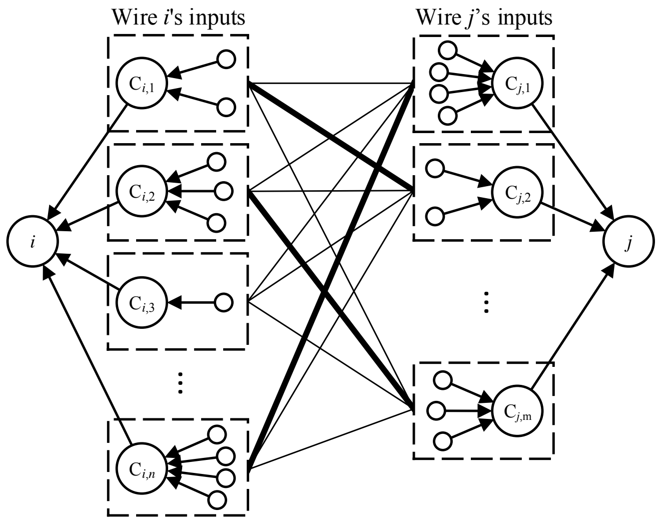


Fig. 3: Conceptual weighted matching of two wires i and j. Thicker lines would represent the best weighted matching.

Algorithm 1 Compute similarity score between two logic vertexes in a $graph$, with indexes i and j, using a given depth, d, of their fan-in subgraphs.

```
 1: function GetSimilarityScore($graph, i, j, d$)
 2:     $max \leftarrow$ Max($graph[i].numChildren$, $graph[j].numChildren$)
 3:     $min \leftarrow$ Min($graph[i].numChildren$, $graph[j].numChildren$)
 4:     if $graph[i].color \neq graph[j].color$ then
 5:         return 0
 6:     end if
 7:     if $d = 0$ then
 8:         return $min$ / $max$
 9:     end if
10:     Let $G$ be a graph with a node for each child of $i$ and $j$
11:     for $a \in graph[i].children$ do
12:         for $b \in graph[j].children$ do
13:             $simScore \leftarrow$ GetSimilarityScore($graph, a, b, d - 1$)
14:             if $simScore \geq Threshold$ then
15:                 Add edge from $a$ to $b$ in $G$
16:             end if
17:         end for
18:     end for
19:     return MaxMatching($G$) / $max$
20: end function
```

d, is also passed into each of these score queries. This depth is reduced by 1 in each recursive function call, and if the current depth is zero, then the return score is the smaller number of children ($min(n, m)$) over the larger number of children ($max(n, m)$) The pseudo-code for RELIC's main procedure is described in Algorithm 1.

The run-time for this algorithm can be easily determined. Each pair of logic vertexes will be checked at most d times. If $Score(i, j, k)$ was already computed, the Dynamic Programming technique of memoization would return the previously computed value. The worst case run-time becomes $d \times N^2 \times O(\text{MaxMatching})$, where N is the total number of logic vertexes in the netlist. Since maximum matching's run-time is polynomial, so is RELIC's run-time, which is one additional advantage RELIC has over traditional graph isomorphism based approaches.

Once the similarity scores are obtained, a simple classification is performed to identify logic registers. Each register

TABLE I: Control logic register identification results on different netlists with varying parameters.

Name	Number of Gates	Number of Registers	Meta-Graph Threshold	Accuracy	Time
MC 8051	6590	578	0.9	89.1	10 sec
RS232	168	59	0.8	79.6	2 sec
32-bit RSA	2139	555	0.8	95.3	3 sec
AES-128	12576	3968	0.8	100.0	4 min

has a counter initialized with zero, and for each similar pair of logic registers (register pairs that have a similarity score above some pre-determined threshold), the logic registers respective counters are updated. Registers with high counters (above some pre-determined value, normally 0) are selected as non-logic affecting registers.

IV. Results

A collection of netlists, including AES, MC8051, RS232, RSA, s349, and AES-128 were used to benchmark the performance of RELIC. For testing purposes RELIC used a depth of 7 on every netlist. With 100% sensitivity of recovering the control logic registers, the overall accuracy was about 90% except in the instance of RS232. Detailed results can be found in Table I and the two low-accuracy cases of RS232 and MC8051 are discussed below. All simulations were run on a desktop of a 3.40 GHz Intel i7-4770 processor.

A. Analysis - MC8051

The MC8051 proved to be very challenging. Due to the size of the fan-in trees a smaller depth was used. This small depth caused many pairs of registers to have higher scores than what they should have. To combat this shift towards a denser Meta-Graph a higher threshold was used. After the parameter changes RELIC using the Meta-Graph heuristic identified 63 registers with the potential to be logic registers. All intended logic registers were in this subset.

B. Analysis - RS232

The RS232 was an example of a netlist that can potentially have poor results when run on RELIC. The most notable problem was that around one third of the registers were classified as being potential state registers. When only one tenth of the total registers are, semantically speaking, state registers. We can try to reduce error by changing the threshold but it still leaves about 20% of the registers being false positives.

This result is believed to have happened due to the structure of the RS232 chip itself. RS232 is the concatenation of two independent modules into one chip. This can cause the registers to be improperly identified as being similar to each other from different modules. These false positives can make the actual similarity harder to detect. This false matching can be prevented by using methods similar to those used in other papers, such as WordRev [6]. Also using both fan-in and fan-out RELIC can reduce the number of false positives and improve accuracy.

V. Conclusion and Future Work

This paper presented a novel polynomial time method for classifying control logic registers and data registers from an arbitrary, and potentially obfuscated, netlist. The significant advantage RELIC has over previously proposed methods for register classification is its ability to use the given netlist as a reference when determining data words. This allows RELIC to bypass most obfuscation techniques and accurately determine and group word registers. By using max-cost-flow and Dynamic Programming RELIC is able to quickly classify medium size netlists. The preliminary results show that this method works on a large number of netlists with different structures and libraries. As a secondary function the given procedure was capable of grouping together logic with similar functions, which can help a user when attempting to determine the full functionality of a chip.

To increase RELIC's run-time efficiency and capability we plan to utilize a representative for distinguishable subgroups. This new method may speed up the current RELIC while generating comparable results and a larger depth can be used on the more complex chip structures to generate a more accurate classification of these registers. Furthermore, we aim to combine this tool with current techniques for determining the full functionality of chips. Tools have been developed to analyze and generate RTL code for netlists, if given the registers that are meaningful to the logic of the chip. The current results produced by RELIC can be used to generate such RTL from netlists, finally allowing end users to verify the trustworthiness of purchased chips or IP cores.

References

[1] Http://www.adnas.com/.

[2] M. Oya, Y. Shi, M. Yanagisawa, and N. Togawa, "A score-based classification method for identifying hardware-trojans at gate-level netlists," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, ser. DATE '15, 2015, pp. 465–470.

[3] X. Zhang and M. Tehranipoor, "Case study: Detecting hardware trojans in third-party digital ip cores," in *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, 2011, pp. 67–70.

[4] F. Koushanfar and A. Mirhoseini, "A unified framework for multimodal submodular integrated circuits trojan detection," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 1, pp. 162–174, 2011.

[5] Http://www.chipworks.com/.

[6] W. Li, A. Gascon, P. Subramanyan, W. Y. Tan, A. Tiwari, S. Malik, N. Shankar, and S. Seshia, "Wordrev: Finding word-level structures in a sea of bit-level gates," in *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, 2013, pp. 67–74.

[7] Y. Shi, C. W. Ting, B.-H. Gwee, and Y. Ren, "A highly efficient method for extracting fsms from flattened gate-level netlist," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 2010, pp. 2610–2613.

[8] T. Meade, S. Zhang, and Y. Jin, "Netlist reverse engineering for high-level functionality reconstruction," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016, pp. 655–660.

[9] K. S. McElvain, "Methods and apparatuses for automatic extraction of finite state machines," U.S. Patent 6 182 268, 2001.

[10] W. Li, Z. Wasson, and S. Seshia, "Reverse engineering circuits using behavioral pattern mining," in *Hardware-Oriented Security and Trust (HOST), 2012 IEEE International Symposium on*, 2012, pp. 83–88.