Advancements in Automated Circuit Grouping for Intellectual Property Trust Analysis

James Inge, Matthew Kwiec, Stephen Baka, John Hallman
Secure Computing & Communications Division
MacAulay-Brown, Inc.
Roanoke, VA 24014 USA
fpga@macb.com

Abstract: *Analysis of intellectual property for malicious functionality often requires understanding of a localized portion in an implemented design to uncover vulnerabilities and hidden functions capable of causing catastrophic effects. Here, we present an approach that has evolved from a manual processes into automation. Automated circuit grouping assembles related circuitry into hierarchical blocks of functions that can be independently analyzed.*

Keywords: Trust; analysis; intellectual property; automation; functional discovery

Introduction

In today's global supply chain, trusting intellectual property that makes up our most critical systems has become an increasing concern. Analysis of this intellectual property for malicious functionality often requires the understanding of a localized portion in an implemented design. Traditionally, this task is difficult and laborious, but sometimes necessary in order to uncover vulnerabilities and hidden functions capable of causing catastrophic effects. Here, we present an approach that has evolved from manual processes into automation. Automated circuit grouping quickly aids an evaluator by assembling related circuitry into hierarchical blocks of functions. They can then analyze the block functions and determine if there is malicious operation present. This allows the evaluator to focus on the real problem at hand, keeping our systems safe and effective for our warfighter.

The unified grouping methodology (UGM) defines how an evaluator should place elements in various hierarchical blocks. It provides a systematic way to handle elements that are on the boundary of a hierarchy and provides for consistent grouping across multiple evaluators. Also for common circuit building blocks, a consistent definition is created. For example, if the block diagram illustrates a block as a "shift register", then the UGM provides a guide for the evaluator to use in determining which logic to include in the "shift register" block. Use of a consistent logic grouping methodology in a reverse engineering flow will result in improved clarity of the final results. This consistency also allows for comparison of two reverse engineering results for the same design, thus enabling grading of a test result against a known good answer. The UGM is easily adaptable to meet the specific needs of the end customer.

The UGM consists of a set of general grouping rules and a set of rules for certain specific circuit building blocks. The general rules cover how to place logic into a hierarchy and how to handle logic that is between hierarchies. There are specific rules for the following building blocks: finite-state-machines (FSM), register banks, RAM/ROMs, FIFOs, shift registers, counters, clock-manipulation logic, clock-domain-crossing (CDC) logic, accumulators, multiplexors, decoders and oscillators.

Methodology

The set of general grouping rules is as follows. In these rules, the term element refers to a design primitive, and grouping is realized by the act of placing elements into netlist hierarchies.

Rule 1. General rules can be overruled by the rules for specific circuit building blocks.

Rule 2. An element can only be a member of a single leaf hierarchy, where a leaf hierarchy is one that contains no sub-hierarchies.

Rule 3. Flops and major macros, such as DSP48 blocks, are assigned to a hierarchy first. Then any combinatorial logic that is uniquely contained between the flops and the major macros is added to that hierarchy

Rule 4. Next any flops between hierarchies, or boundary flops, are assigned to a hierarchy using the following sub-rules:

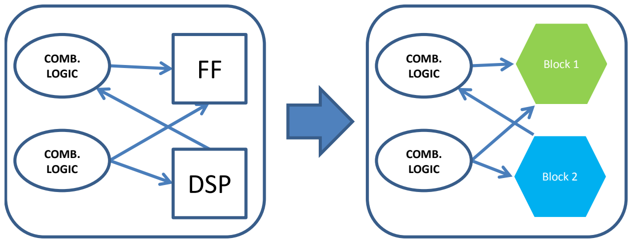
> ***4a***. Boundary flops are preferentially placed on hierarchy outputs. Therefore, if there is a single boundary flop on a signal, then that flop is assigned to the hierarchy where it would be considered an output.
>
> ***4b***. If there are two boundary flops on a signal, then the two flops are split between hierarchies.

Rule 5. Next any combinatorial logic between hierarchies, or boundary logic, is assigned to a hierarchy using the rule: If-and-only-if an element drives a single hierarchy, then that element is included in that hierarchy.

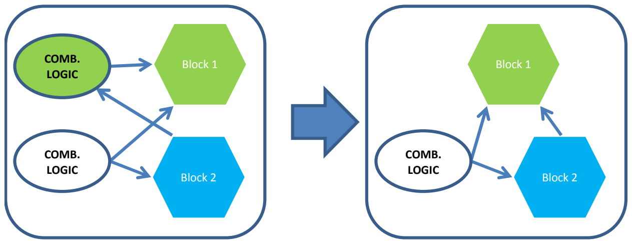
Rule 6. Any element that is left unassigned is assigned to the top-level hierarchy. For example, common "glue" logic that drives several modules would be assigned to the top-level hierarchy.

As an example of this approach, consider the grouping of stateful logic blocks. In this case, all of the elements inside the circuit are initially classified as either sequential or combinational and the following steps illustrate the process of iteratively creating hierarchical blocks that combine related state and state control logic.

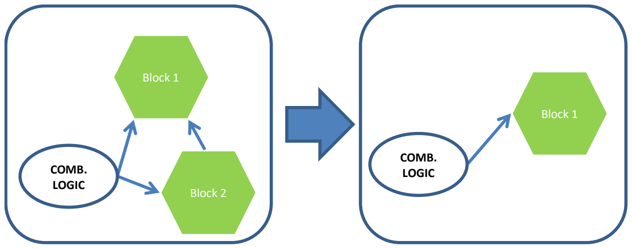
1. Assign all sequential elements in the circuit into their own hierarchical block.



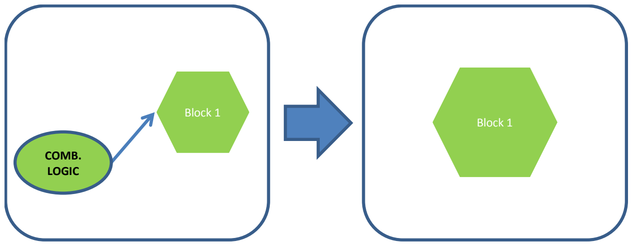
2. Trace back each sequential hierarchical block's inputs and combine all combinational logic that uniquely feeds that hierarchy into the hierarchical block.



3. Trace back each hierarchical block's inputs and combine all hierarchies that only feed another hierarchical block into the tracing block.



4. Next assign any combinatorial logic between hierarchical blocks, or *boundary logic*, to a hierarchy using the rule: If-and-only-if an element drives a single block, then that element is included in that hierarchy.



5. Steps 3 and 4 are repeated until there is no more logic to combine.

6. Any created hierarchies that only contain the original sequential element are disbanded and placed back into their original hierarchy.

Experimental Setup and Results

To test the UGM we used a small System-on-Programmable Chip (SoPC) design. The design is a Wishbone rev B.4 compatible IP core designed to be inserted in a Xilinx Virtex-5 FPGA computing environment. The SoPC IP core encompasses several other home-grown and 3rd party IP modules hanging off a common 32-bit Wishbone slave interface. These cores include an OpenCores I2C Master module, a custom on-chip memory module, a custom arithmetic logic unit module, and a custom Ethernet frame check sequence generator module. Though not incredibly large, the design is sufficient to demonstrate the improvements in the hierarchy reconstruction of the flattened design.

The UGM algorithm was run with its combine-unique-node-limit set to 5, which limits any generated hierarchies to a minimum of five components (i.e., gates, flops, LUTs, etc.). The tool was run on two variants of the SOPC design. The first was a raw netlist generated directly from the FPGA bitstream and contained roughly 7.4K components. We then processed this netlist via some in-house-developed circuit normalization functions to produce a second design containing a minimal amount of synthesis artifacts (~5.1K components) to better evaluate the impact of circuit chaff produced by the forward-flow FPGA implementation tools. The normalize functions both optimize and simplify the design, but do not change the function of the original design. Both the raw and the normalized netlists were completely flat (i.e., contained no hierarchy) prior to applying our UGM.

The results in both variants of the SOPC design showed promise, as shown in Table 1. In both cases the UGM algorithm encapsulated over 40% of the design into new hierarchies with modules ranging in size from a handful to hundreds of components. Thus, even without netlist normalization, the UGM process significantly reduced the final circuit complexity. With the netlist normalization pre-processing step, however, over half the circuit components were encapsulated into some hierarchy and they were much more uniform in size.

Table 1: Components in Hierarchies

Netlist	Total Comp.	Hierarchies Created	Encap. Comp.	Remaining Comp. (top)	Largest Hierarchy
Raw	7407 (100%)	101	4365 (41%)	3042 (59%)	1709 (23%)
Normal-ized	5094 (100%)	109	2650 (52%)	2,444 (48%)	370 (7%)

In another perspective, we examined the size of the hierarchies created as shown in the table and distribution figures below. The normalized case shows that a significantly higher number of hierarchies were created with the medium range number of components. These larger blocks, though still relatively small, provide confidence that these blocks can be combined further and remain at a size understood by design and verification engineers. This brings us closer yet to automatically recreating hierarchies similar to the original design.

Table 2: Number of Hierarchies with N Components

Netlist	< 10	10-100	100-1000	1000+
Raw	40	49	1	1
Normalized	31	73	5	0

Figure 1: Module Size Distributions: Raw Netlist

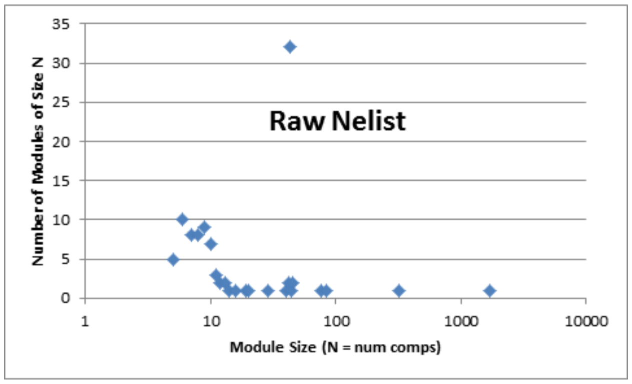
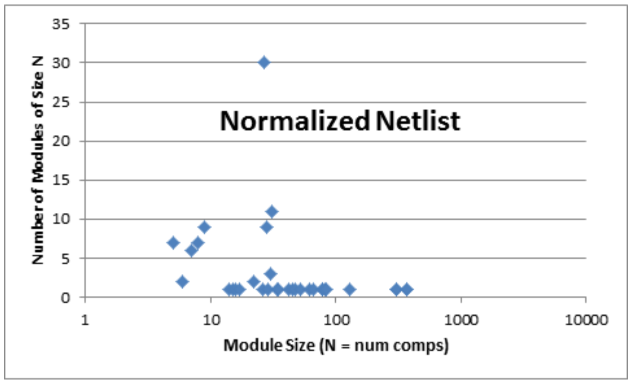


Figure 1: Module Size Distributions: Raw Netlist



Since the SOPC was a custom design, we had access to the clear-text version of both netlists and were able to perform a qualitative evaluation of the results. In general, the algorithm was particularly good at identifying flag/status indicators (e.g., full, empty, result_is_zero), interrupt generators, serial data paths (e.g., I2C), complex state bits in FSMs, and especially areas where the data path contracted in width (e.g., read-back data from addressable registers or memory). In fact the largest hierarchy created in both netlists was the Wishbone read-back logic shared by all IP in the SOPC; which consists of multiple layers of multiplexing and registering. Regions of the design that remained largely untouched by the algorithm included any counters, such as FIFO pointers and the hardware-enforced IP licensing mechanism inserted for previous test purposes. Also not encapsulated was much of wishbone fabric itself – particularly the control logic and less complex FSM state bits.

Future Work and Conclusions

With the unified grouping methodology, we have defined a common set of guidelines that provides the framework for an iterative and repeatable combining process. We have now automated the process and the result produces a design grouped into hierarchical blocks. When applied to intellectual property, this automation provides an efficient starting point for an evaluator to begin a trust assessment. While these experimental results and early use in practice have shown noticeable improvements, we have already identified some potential future enhancements and some operational procedures to yield optimal results. In future work, we plan to explore enhancements that provide usability and flexibility as well as improve quality of results. Some of these possible options include:

- Execution on the entire design (including existing hierarchies) or only a specified hierarchy
- Execution on a specified list of elements
- Perform multi-pass (iterative) operation

What had often taken weeks of manual effort has now been reduced to an overnight process or just a matter of hours. This new starting point enables the evaluator to direct focus on the real problem at hand, keeping our critical systems free from malicious operation.

References

1. T. Sobh, M.K. Elleithy, S. Patel, "Reverse Engineering of VLSI Chips: A Roadmap," Journal of Engineering and Applied Sciences 2, pp 290-298, 2007
2. DARPA Integrity and Reliability of Integrated Circuits (IRIS) program, 2011 – 2013, http://www.darpa.mil/program/integrity-and-reliability-of-integrated-circuits
3. DARPA Trust in Integrated Circuits (TRUST) program, 2008 – 2011, http://www.darpa.mil/program/trusted-integrated-circuits
4. J. Graf, S. Harper, L. Lerner, "Ensuring Design Integrity through Analysis of FPGA Bitstreams and IP Cores," The 2012 International Conference on Engineering of Reconfigurable systems and Algorithms (ERSA'12), Las Vegas, NV, July 2012.