Leveraging FPGA Primitives to Improve Word Reconstruction during Netlist Reverse Engineering

McKendrick, Reilly Simpson, Corey Nelson, Brent Goeders, Jeffrey

***Abstract*—While attempting to perform hardware trojan detection, or other low-level design analyses, it is often necessary to inspect and understand the gate-level netlist of an implemented hardware design. Unfortunately this process is challenging, as at the physical level, the design does not contain any hierarchy, net names, or word groupings. Previous work has shown how gate-level netlists can be analyzed to restore high-level circuit structures, including reconstructing multi-bit signals, which aids a user in understanding the behavior of the design.**

In this work we explore improvements to the word reconstruction process, specific to FPGA platforms. We demonstrate how hard-block primitives in a design (carry chains, block memories, multipliers) can be leveraged to better predict which signals belong to the same words in the original design. Our technique is evaluated using the VTR benchmarks, synthesized for a 7-series Xilinx FPGA, and the results are compared to DANA, a known word reconstruction tool.

I. Introduction

Understanding a low-level hardware netlist, and reverse engineering the original circuit design, functionality, and intent, is essential for several different design applications. While such a processes can be used maliciously (e.g., to steal IP, plan Hardware Trojans attacks), it also is essential for many "good" applications, including Hardware Trojan detection, confirmation of IP correctness, equivalence checking, detecting IP infringement, obsolete product analysis, and more [1], [2].

Unfortunately, uncovering the high-level intent and design of a gate-level netlist is challenging. The netlist is typically a sea-of-gates, containing only library or device primitives, without any module hierarchy, distinct control structures (i.e. FSMs), groupings of signals or flip-flips into words, or larger arithmetic units. Netlists are typically extracted from the physical implementation; for ASICs this involves analyzing GDSII files or delayering circuits, while for FPGAs, several projects exist that have documented commercial bitstream formats and offer bitstream-to-netlist tools [3]. Given this, the netlists are absent of any net names, further challenging the process of understanding the higher-level design structure.

Several previous works have presented approaches to reconstruct original circuit structure, including identifying known modules from a library [4], locating commercial IP [5], and locating register files, counters, adders, and subtractors [6]. One component of rebuilding the original design, and the focus of this work, is *word reconstruction*, which attempts to recreate multi-bit signals and registers from individual nets and flip-flops in the netlist [7], [8].

In this short paper we propose and evaluate an enhancement to the word reconstruction process, specifically when targeting FPGA netlists. Previous work [7], [8] has not leveraged architecture-specific primitives, and the existing algorithms only consider flip-flops and their connection patterns. In this work we propose leveraging FPGA hard-blocks (adders, memories, multipliers) to seed the iterative word reconstruction process, as they typically contain multi-bit signals.

We evaluate our approach on the full VTR7 FPGA benchmark set, and demonstrate that our enhanced approach significantly improves upon the word reconstruction result (using the same QoR metrics from existing work). In several benchmarks where existing work fails to provide a good word reconstruction (ie below 80% score), we are able to improve the result substantially, from 0.71 to 0.91, and for many other benchmarks where existing work already provides a good solution, we maintain the same, or slightly better, QoR.

II. Background

There have been two major works that have presented approaches for word reconstruction, WordRev [7] and DANA [8].

WordRev, created by Li et al. [7], groups nets together to identify *functional* words, looking for cones of logic with matching boolean equations. They then iteratively propagate these results through the netlist, considering all possible word groupings. The approach is compute intensive, and the explored solution space grows rapidly, resulting in potentially long runtimes. The WordRev work tests their tool against several designs, providing runtime values, but does not provide any quantitative results or metrics to evaluate the accuracy of their groupings. In addition, the WordRev tool is not publicly released, so there is no straightforward way to compare WordRev against our proposed approach.

DANA, created by Albartus et al. [8], is another tool designed to perform word reconstruction; however, this tool takes a *structural* approach. DANA analyzes connection patterns to try and estimate which nets should be grouped into a word. This is done iteratively, by seeding candidate groupings and then iterating forward and backward through the circuit, continuously grouping signals by whether they connect to common signals in other candidate groups. Multiple different passes are performed, and a final voter chooses between different potential groupings, giving precedence to common groupings and larger cluster sizes. DANA also demonstrates that by analyzing and understanding the dataflow through the circuit, one can more easily detect larger circuit structures. For example, after register reconstruction they look for registers that are 256-bits in size, which enables them to locate the boundaries of an AES256 module in the design. They in turn use this to locate hardware Trojans by looking for suspicious

circuit patterns, such as unexpected data leaking out of the encryption module. In evaluating their word reconstruction algorithm, the DANA authors proposes two metrics, *purity* and *normalized mutual information (NMI)* (described in more detail in Section IV-A), to measure how accurately register groupings match the golden netlist.

When analyzing the results of word reconstruction algorithms, it is important to recognize that there is no "perfect algorithm", that could always create word groupings exactly matching the original netlist. When a designer writes RTL, they can make arbitrary decisions about how signals are grouped into words, and may choose to write RTL where different bits of a word have different functional or structural behavior. For example, when designing a CPU in RTL, the entire instruction could be grouped as a single register, or it could be split into one register for the opcode, and one for each opcode argument. Both RTL styles would produce the same netlist, but with different word groupings, meaning there is a many-to-one relationship between RTL and the resulting netlist (i.e. there is information loss in the synthesis process). Despite this shortcoming, we lack a better way to evaluate word groupings, and have elected to follow the previous DANA work in evaluating the correctness of word groupings against the named register groupings from the netlist.

III. Solution Overview

In our proposed approach, we leverage FPGA-specific primitives to increase the accuracy of structural word reconstruction. We demonstrate this approach on a Xilinx 7-series architecture, leveraging hard-block adders, block RAMs and multiply-accumulate units (CARRY4, RAMB18E1, RAMB36E1, DSP48E1). While we target this FPGA family, the techniques can easily be expanded to other primitives and implemented with other vendors and families.

A. Leveraging FPGA Hard-Blocks

Similar to DANA, our implementation of a word reconstruction algorithm begins by abstracting the FPGA netlist into a weighted, directed graph, where registers are represented as vertices and all combinational logic between the registers is removed and represented as simple edges. Unlike DANA, the aforementioned FPGA hard-block primitives (adder, memory, multiplier) are also preserved as vertices. Each edge is annotated with a boolean value, indicating *true* if the edge is a direct wire connection, or *false* if combinational logic or other device primitives were absorbed into the edge. The primitives IBUF, OBUF, IOBUF, BUFG, VCC, and GND are also preserved to keep track of design I/Os and constants.

Next, the hard-block vertices are selected and their connections are examined, and registers connected to the primitive's I/O buses can be inferred to be belonging to the same word. In the case of CARRY4, signals connecting to the *DI* and *S* 4-bit inputs, as well as the 4-bit *O* output, can each be considered to be part of the same multi-bit word, since this primitive is typically used to sum two multi-bit values together, representing the operation of $DI + S = O$. As such, when a CARRY4 is encountered, and the edge is marked *true* (a direct wire connection), then the registers can be grouped together with high confidence.

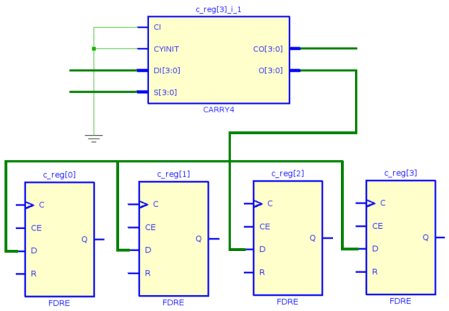


Fig. 1: CARRY4 block connection example

As an example of this process, consider Figure 1, where the *O* output pins each directly connect to a FDRE flip-flop. In this case, the tool would correctly group these registers together, effectively recognizing the 4-bit word *c_reg*.

When the edge is *false*, then some intermediate combinational operations exist between the output pins and the flip-flops, and the signals cannot be automatically grouped together. For example, during graph construction, if a LUT6 were encountered in the netlist, then six *false* edges would be added to the graph connecting input to output nodes. In such a case where you have multiple bits influencing a single output register, one cannot be certain exactly how the registers should be grouped together, and so these edges are ignored at this stage of the algorithm.

Like the CARRY4, the BRAM and DSP pins can be similarly analyzed to infer word groupings. Of specific interest are the data ports of the BRAM and DSP as well as the read/write address ports of the BRAM.

B. Cascaded Hard-Blocks

In some designs, primitives are cascaded to allow for their operation on larger word values. For example, this is commonly done with the CARRY4, where the *carry out* value of one adder is fed into the *carry in* pin of the next CARRY4 in the chain. BRAMs also have an internal cascade option allowing multiple memories to be combined together, into one larger address space. When cascaded primitives are encountered, they can be combined and abstracted into one large primitive, which will allow the subsequent register reconstruction algorithm to be seeded with even larger register groupings.

C. Word Reconstruction

Once the hard-block primitives are all analyzed, they are removed from the graph structure. Next, our word reconstruction algorithm proceeds to establish word groupings, using a very similar algorithm to DANA.

In our implementation, vertices are first grouped into stages based on logic depth to the nearest output pin. The algorithm then iterates through the stages, examining each register. The input connections to the register are collected and their cluster

status is valued. The value is a string that represents if any of the connected vertices are clustered, and what clusters are present. Then, all registers in the stage are compared and those with similar values are marked as belonging to the same cluster. The outputs of the stage are then analyzed in a similar way. The algorithm continues like this stage by stage until the design inputs are reached.

This technique is very similar to DANA, except that they perform multiple passes with different starting conditions, and use a voter to select from different candidate word groupings. Our approach utilizes the primitive information to seed groupings, so it doesn't make as much sense to explore different starting conditions (in addition, the DANA paper mentions that there are marginal benefits with increased number of passes).

IV. Results

A. Evaluation Metrics

The QoR of the register clustering is determined by comparing the resultant register clusters with the actual clusters from the netlist, where registers with the same name in the netlist are assumed to be the same word. The authors of DANA used two metrics to evaluate cluster quality: *Purity* and *Normalized Mutual Information (NMI)* [8]. Both metrics are a number between 0.0 and 1.0, with 1.0 being a perfect score [9].

Purity evaluates individual cluster accuracy. Each resultant cluster is marked as correlating to a word from the golden netlist, depending on which word most frequently appears in the cluster. The purity value of this cluster is calculated as the percentage of signals in the cluster that actually belong to this word from the golden netlist. The score of all clusters is then averaged together. However, Purity alone is an imperfect metric since purity is 1.00 when every register bit is assigned to its own cluster [9], essentially rewarding no clustering effort.

NMI is a technique to compare two partitionings, and is based on the *Mutual Information* concept from information theory [9]. NMI compares cluster quality with the whole solution space. Unlike purity, the NMI score is influenced by whether the quantity of proposed clusters equals the quantity of solution clusters, whether the general sizes of the proposed clusters equals the sizes of the solution clusters, and whether the content inside each cluster is equal to the solution clusters. The only way to get a perfect score with the NMI metric is to group the clusters exactly the same as the solution clusters [9].

To summarize, low NMI can represent an inaccurate total number of clusters, bad cluster sizing, or registers incorrectly grouped together. Low purity is only caused by registers incorrectly grouped together. Higher purity can be manipulated by restricting cluster size to be very small, but this will cause the NMI score to be low since cluster sizes/quantity does not accurately represent the solution space. A higher NMI is not necessarily achieved by forcing larger clusters, since NMI, like purity, is negatively influenced by incorrect register associations. Purity can help to understand the NMI metric. For instance, high purity but low NMI implies that registers are clustered together well, but the total number of registers and/or register sizes is inaccurate. The NMI score would likely be improved by combining some clusters together. A high NMI with low purity means quantity/size of clusters is a fairly accurate representation of the solution, but the individual registers in the grouping are poorly chosen.

	NMI			**Purity**		
	Min	*Max*	*Avg*	*Min*	*Max*	*Avg*
DANA [8]	0.643	0.968	0.803	0.134	1.000	0.641
Our Hier. Pass	0.000	0.975	0.800	0.145	0.992	0.590
w/ Primitives	0.803	1.000	0.917	0.580	1.000	0.816

TABLE I: Summary of Results

B. Benchmarks

Both WordRev [7] and DANA [8] test their approach using selected designs from OpenCores.org. While the DANA tool (and our work, without the hard-block optimization) achieve an average score greater than 0.9 for these designs, most benchmarks thus tested are fairly small, and use few FPGA primitives. When we tested the tools with larger and more varied designs, we found that average scores were often much lower.

Given this, we elected to evaluate our approach using the VTR7 [10] benchmark suite, which contains larger, more varied designs. In addition, these designs "come from a variety of real applications" [10] and are commonly used in the FPGA research community. Larger designs do increase the runtime of the algorithm, but all of the designs took less than 10 minutes to analyze.

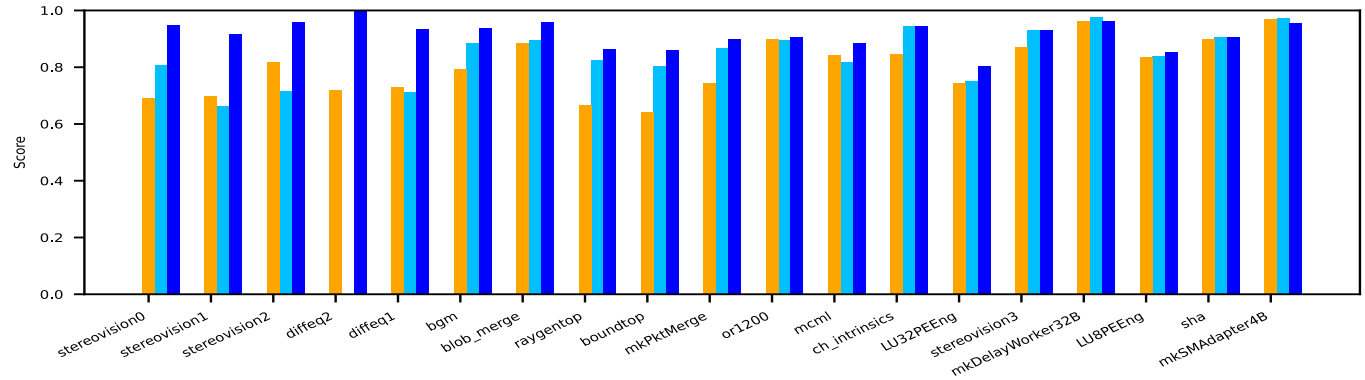
C. Purity and NMI Results

Table I shows that DANA achieves an average NMI score of 0.80 and an average purity score of 0.64 on the VTR benchmarks. Our project's hierarchical analysis, performed *without* our proposed primitive analysis, achieves similar results, with an average NMI score of 0.80 and an average purity score of 0.59. When we include our proposed improvement, leveraging FPGA hard-blocks, the NMI increases to 0.92 (14.2% improvement) and purity to 0.82 (27.3% improvement). Our work was particularly helpful in cases where the original algorithm struggled: the minimum NMI score was raised from 0.64 to 0.80, and minimum purity from 0.13 to 0.58.

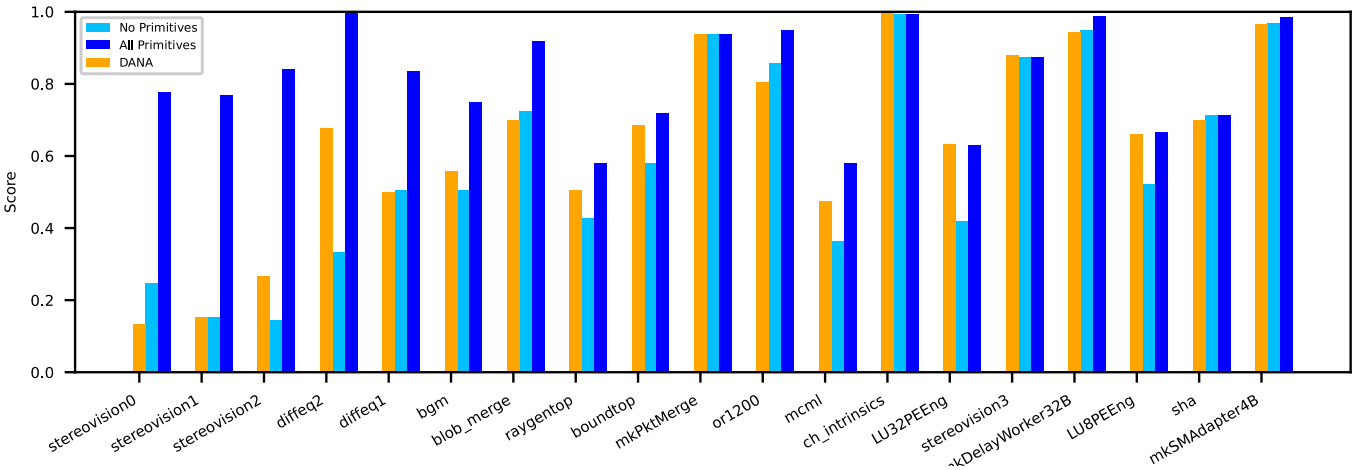
Figure 2 shows the QoR scores for each of the benchmarks. Each benchmark has a bar representing DANA's score, and this project's analysis with and without the hard-block primitive optimization. Note that there are no cases where our enhancement causes a significant drop in quality of the result. In two cases, there is a marginal drop in NMI score; however, this drop is less than 2% and in both cases the purity score is raised by 2-4%. Many benchmarks saw significant improvement in both scores, with 12 designs improving by at least 20% in either NMI or Purity.

D. Analysis

We analyzed the register groupings with and without our hard-block primitive pass and found that the primitive pass had



(a) NMI



(b) Purity

Fig. 2: NMI and Purity scores for DANA, and our work, without and with our hard-block primitive pass.

significant impact on the result; on average, 50% of register groupings were modified in some way. It is worth noting that the hard-block primitive pass does more than just group registers connected directly to primitives, since these results are used to seed the hierarchical analysis, their influence can be far reaching. This means the results from the primitive pass can have a butterfly effect on the rest of the analysis.

We investigated how the number of hard-blocks in the design affect Purity and NMI scores, and found that a high number does not necessarily guarantee large improvement. For instance, in *blob merge* there is a 2:1 ratio between CARRY4s and registers (in fact, it is the only design in the benchmark suite where the hard-block primitives outnumber the registers), and while the achieved NMI score is quite high, it is still fairly good without the primitive pass. We noticed that in many cases a CARRY4's output feeds directly into the inputs of other CARRY4 blocks; with no registers in-between, there is not much improvement to be made.

Other factors that can negate the advantage brought by the primitives is the complexity of the design. The words in simple designs can be deduced rather easily, and thus the words seeded by the primitive analysis have little influence. Slight improvement is still possible, since the primitive analysis causes a pre-grouping to occur that affects the propagation of words in the hierarchical stage.

Naturally, the two designs with the least number of hard-block primitives per register, *stereovision3* and *ch_intrinsics*, showed zero improvement.

V. Conclusion

Utilizing FPGA hard-blocks in word reconstruction provides improvement over existing hierarchical analyses algorithms. While this project demonstrates its effectiveness on primitives found in Xilinx 7-series chips, the methods can be expanded to other hard-blocks, like shift registers, and to other vendors.

References

[1] S. E. Quadir, J. Chen, D. Forte, *et al.*, "A survey on chip to system reverse engineering," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 13, no. 1, 6:1–6:34, Apr. 13, 2016.

[2] E. Cahill, B. Hutchings, and J. Goeders, "Approaches for FPGA design assurance," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 15, no. 3, 28:1–28:29, Dec. 28, 2022.

[3] H. Yu, H. Lee, S. Lee, Y. Kim, and H.-M. Lee, "Recent advances in FPGA reverse engineering," *Electronics*, vol. 7, no. 10, p. 246, Oct. 2018.

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

[5] C. Simpson, "Towards trojan detection from a raw bitstream," *Theses and Dissertations*, Mar. 23, 2022.

[6] P. Subramanyan, N. Tsiskaridze, W. Li, *et al.*, "Reverse engineering digital circuits using structural and functional analyses," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 1, pp. 63–80, Mar. 2014.

[7] W. Li, A. Gascon, P. Subramanyan, *et al.*, "WordRev: Finding word-level structures in a sea of bit-level gates," in *International Symposium on Hardware-Oriented Security and Trust (HOST)*, Jun. 2013, pp. 67–74.

[8] N. Albartus, M. Hoffmann, S. Temme, L. Azriel, and C. Paar, "DANA universal dataflow analysis for gate-level netlist reverse engineering," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 309–336, Aug. 26, 2020.

[9] C. D. Manning, P. Raghavan, and H. Schütze, "Evaluation of clustering," in *Introduction to Information Retrieval*, Cambridge University Press, 2008.

[10] J. Luu, J. Goeders, M. Wainberg, *et al.*, "VTR 7.0: Next generation architecture and CAD system for FPGAs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, no. 2, pp. 1–30, Jun. 2014.