Discovering Reusable Hardware Designs Using Birthmarking Techniques

Kevin Zeng
Bradley Department of
Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA
Email: kaiwen@vt.edu

Peter Athanas
Bradley Department of
Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA
Email: athanas@vt.edu

***Abstract*—With decades of investments, standardized hardware design practices have not taken root. Productivity has continued to be a major issue for many designers. One of the most important factors in improving productivity is design reuse, which current tools have done little to address. Reusing designs involve manually searching through repositories to find a design that meets the requirements of a project. This search can be very tedious and often unfruitful. In order to promote design reuse, an automated discovery technique is proposed: a reference circuit is compared with an archive of existing designs such that similar circuits are suggested throughout the design phase; however, most circuit comparison techniques are focused on exact matches. In this paper, a method of comparing the similarity of circuits using a birthmarking technique is presented. The birthmarking technique captures a wide range of features including structural characteristics, such as components and nets, and functional characteristics, such as dataflow, into a single compact representation. Experiments and evaluations of the birthmarking technique were performed on over 150 circuits from various sources in order to show the feasibility of the proposed methods. Results show that using birthmarks to compare the similarity of circuits is promising.**

I. Introduction

It is highly possible that most of the components that are commonly used in digital logic have already been designed in one form or another. If the designs can be reused, there is no need to needlessly reinvent the components again. Time spent during the redesign can be allocated to more important tasks of the design phase. The very reason domain-specific software libraries exist and have been extremely successful is to provide the user with a set of commonly used functions that can be imported and used immediately, increasing the overall productivity of the designer. In terms of reusable designs, it is assumed that they have an underlying expectation of functional correctness. This means that reusable designs have been tested and are expected to function correctly as is. Due to the underlying expectation of functional correctness for reusable designs, most of the simulation and verification of the circuit is eliminated. All that is left is the integration of the component into the overall design. Table I shows the overall cost breakdown of a typical reference design for military applications [1]. The simulation, verification, and implementation phase of a design takes up the largest portion of the design phase which reusing designs can help greatly alleviate. Therefore, reusing instead of reinventing is one factor that can lead to a significant increase in productivity of designing hardware.

This work was supported in part by the I/UCRC Program of the National Science Foundation within the NSF Center for High-Performance Reconfigurable Computing (CHREC), Grant No. IIP-0804155.

In spite of these benefits, the reuse of past designs has not gained significant traction in the hardware community. Even if reuse is seen in practice, limitations such as knowledge of the components in the library, inconsistent documentations, and the search for the similar components hinder the full potential of reuse. Vendors have come a long way in terms of increasing productivity by providing powerful tools for synthesis, debug and simulation, and placement and routing [1] [2]. High-level synthesis (HLS) and programming languages, such as OpenCL, allow users to write their own accelerator with the underlying details of the hardware abstracted away. Nonetheless, these tools have done little to incorporate design reuse into the design methodology. Many tools allow the import of third-party Intellectual Property (IP) cores into the core library through which they can be searched, but that is the extent of reuse seen in most vendor tools.

Though the contemporary tools simplifies reuse to an extent for the users, they still have many drawbacks. Users have to import specific IP cores they want to reuse. This would first entail the search across various sources for existing components that fits the requirements of the design as well as the necessary documentation behind the IP. Reusable hardware libraries or components themselves are not as prevalent either, making the search for relevant IPs difficult. Hardware repositories can

TABLE I
Cost of a typical reference design for FPGA

Development	Hours
Architectural Design	1886.9
Detailed Design	4745.1
Simulation, Verification, and Implementation	5376.3
Place and Route	1809.2

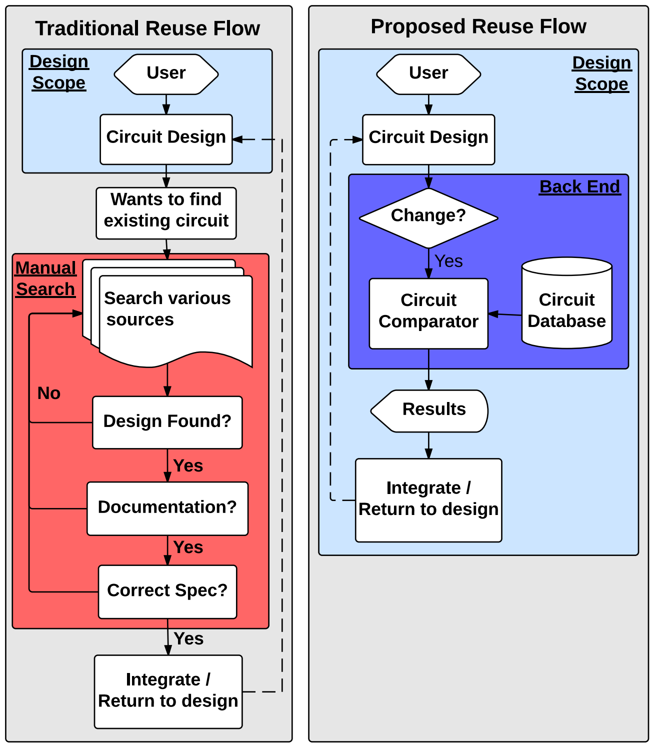

Fig. 1. Comparison between the proposed flow and the traditional flow for reusing designs

often be incomplete and disorganized. A complete and private library within an organization can have certain limitations as the users need to familiarize themselves with what the database contains. If there exists a tool where the candidate cores from a given database are automatically suggested to the user throughout the design phase, then designers will be more open in accepting and reusing components. By extension, if the above assertion is true, the *design* is transformed into a *discovery* process.

Figure 1 displays the high-level concept of the proposed model. As the user makes changes to a reference circuit, the proposed tool will continuously monitor the circuit during the entry process. A vast number of comparisons are performed between the emerging design and an archive of existing designs. If there exists a proper community structure of easing contributions and automating access, the number of existing designs can rapidly rise to tens of thousands. Candidate circuits are then suggested to the user within the design environment in an unobtrusive manner. The results are ranked based on similarity such that most relevant information is presented to the user first. The tool can then suggest additional similar circuits, display relevant documentation on the design, or import the existing circuit into the design for the user to use. Automatic integration of the existing design to the current can be a potential feature as well.

Traditional reuse flow requires the user to manually search for designs they want to reuse which can be very time consuming and often leads to many dead ends. The proposed flow takes away the manual search the user has to perform. In addition, because the idea of reuse is integrated within the design environment, the designer can remain in the scope of the project and as a result, remain focused on the design and task at hand. By automatically suggesting existing designs to the user, design reuse becomes more appealing, requiring little to no effort from the user.

In order to predict relevant designs the user may want reuse, comparisons between a reference circuit and a database of circuits need to be performed. Much of the research for comparing circuits is to find exact matches by using graph-based approaches to compare the netlist of two circuits [3] [4]; however, these methods usually only consider the structural aspects of the circuits and neglect the functional aspects. Structurally different circuits can be functionally equivalent. The data width can also be an issue. As the data width increases, the structure of the circuit gets more complex even though the overall functional remains the same. Functional comparison of circuits have been focused on combinational equivalence checking (CEC) to determine if two circuits exhibit the same behavior. Several techniques are commonly used for CEC such as binary decision diagrams (BDDs), or satisfiability solvers (SAT). Still, these approaches have limitations such as input variable ordering and input output correspondence between two designs. These approaches also focus on finding an exact functional match since the goal of these approaches is correctness and verification.

In this paper, a new methodology for comparing the similarity of two digital circuit using a *birthmarking* approach is proposed. A birthmark is defined by the inherent characteristics of a circuit. The birthmark is constructed so that the structural and functional information of the circuit are accounted for. The interest in comparing circuits is not to find an exact match, but a metric that indicates both the structural and functional similarities between two circuits. In order to validate the proposed birthmarking method, an assessment is performed on the quality of the results returned when comparing a reference circuit against a compiled database of open-source IP cores from various archives.

II. Related Work

In terms of circuit comparison, there has been little work done in determining the *similarity* between two circuits. Shi et al [5] used a modified version of an iterative graph similarity algorithm to find similar nodes in a reference and modified design in order to improve placement on FPGAs. The approach focused on the similarity between each of the nodes rather than the overall design. InVerS [6] determines a similarity factor between two netlists based on signatures of the nets obtained using a fast simulation technique. Since InVerS can characterize equivalent circuits as dissimilar, this technique is focused more as an incremental verification method.

Most of the applications regarding circuit comparison requires the search for an exact match both structurally or functionally. Technology mapping of primitive components requires an exact subgraph match such that the subgraph

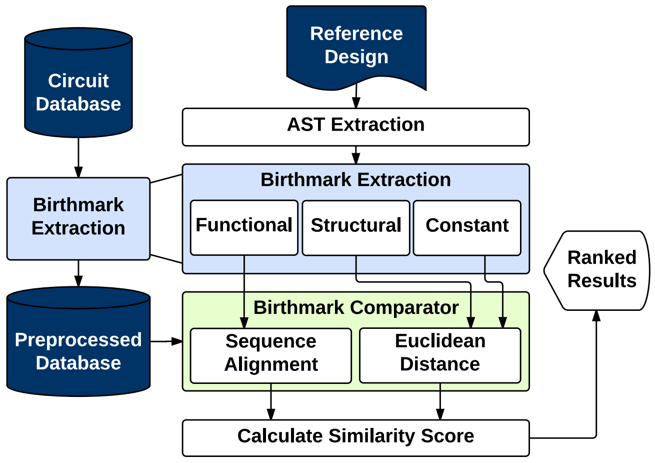

Fig. 2. Block diagram of overall system flow

is replaced with a library primitive. Layout vs Schematic (LVS) checking looks at the circuit's layout and schematic to make sure the two representations are an exact match. Subgemini [4] is a tool that uses subgraph isomorphism techniques to find subcircuits within larger circuits. Whitam et al [3] found similar designs in a circuit repository using a subgraph isomorphism technique. Their approach is similar to Subgemini yet limited as its focus exact structural matches of analog circuits. The previous approach [7] to this problem explored different graph-based algorithms for structural similarity matching such as a maximum common subgraph. The problem with many of these techniques is that most graph algorithms are NP-complete and would be infeasible for large and complex circuits. The proposed system needs to perform many comparisons quickly and efficiently for circuits of various sizes and complexity. Otherwise, by the time similar circuits are suggested, the reference design could be in a completely different state than when it was last examined.

Aside from structural comparison, there has been existing work in trying to compare circuits based on their functionality. Subramanyan et al [8]. extracts high-level components from flattened netlists by partitioning the circuit into k-cuts. The function of the k-cuts are then compared to a database of pre-defined bitslices. Yet, in order to be computationally feasible, the inputs are of the k-cuts are limited to six. Furthermore, depending on how structural netlist is arranged, the function of the cut can be obfuscated due to side inputs. Li et al [9] tackles the same problem but using formal verification techniques. However, the formal properties of the circuit has to be manually described and cannot be extracted from the circuit description itself. Many of these endeavors do not give a good sense of how similar two circuits are.

III. Comparing Similarity of Circuits

In order to suggest reusable designs to the user, the main issue that needs to be addressed is how to assess and compare the similarity between two circuits. Comparing the similarity of circuits is a difficult task. Hardware designs have complex structures and functions that can be represented in countless different ways. Also, different designers may have different definitions of what is considered similar. For example, how similar is a multiplier to a shift operation? Both are fundamentally different, and yet, many hardware designers use the left shift operator to perform very fast and efficient multiplications by two. Based on previous work, this is the first paper that explores an approach to capture the underlying nature of the circuit in a compact representation such that the similarity of two circuits can be compared. The flow of the proposed method is depicted in Figure 2.

A. System Overview

Starting from a reference design, the abstract syntax tree (AST) is extracted. In this paper, a textual design entry with a hardware description language (HDL) such as Verilog or VHDL is used, but can be extended to various design entry platforms such as graphical entry tools. The previous approach to this problem [7] explored using information from the flattened netlist as a starting point. While a netlist contains a common representation among all designs, it is difficult to extract meaningful data efficiently especially for large and complex circuits. In addition, important information such as hierarchy is lost during the flattening process. The AST on the other hand contains a compact and descriptive representation of the overall design. Once the AST is extracted, the different components of the birthmark, defined in the following section, can be constructed. The birthmark is passed to a central server, which compares it against the birthmarks of existing designs to determine which circuit is most similar. Results of the comparison are then sent back to the client in a ranked order so that the user is aware of which designs are most relevant to the reference.

B. Hardware Birthmarking

A hardware *birthmark* is defined as a representation of the overall structure and functionality inherent to a specific circuit. Birthmarks are commonly seen in software [10], for detecting plagiarism [11], malware and viruses [12] [13], and theft [14]. The overall idea of a birthmark is leveraged and extended for hardware designs, which is described further in this section. The similarity of the circuit can be assessed by directly comparing the birthmark themselves.

The general hardware birthmark scheme of a given circuit is partitioned into three different components: the functional component, the structural component, and the constant component.

1) Functional Component: The first component of the birthmark targets the AST or dataflow DF of a circuit C. A dataflow is a directed graph $DF = (V, E)$ of C, with a set of inputs $I \subset V$ and outputs $O \subset V$, where $|I| \geq 1$ and $|O| \geq 1$. The dataflow of a 8-bit up counter can be seen in Figure 3. From the dataflow, functional information of a circuit can be extracted in the form of a sequence where the sequence represents a datapath. A datapath DP is a path, $\{v_1...v_m\}$, in DF where $v_1 \in I$ is the source node and $v_m \in O$ is the

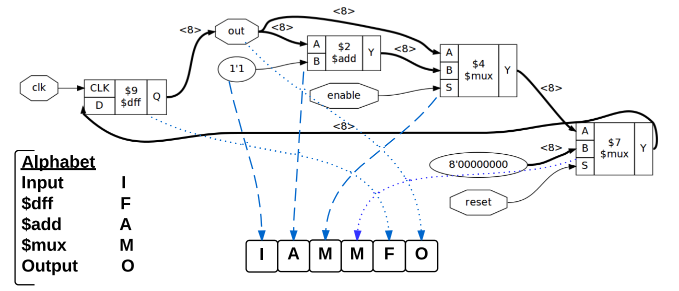

Fig. 3. Datapath is extracted from the dataflow from an input port to an output port. Each operation is assigned a letter in an alphabet to for a sequence

sink node. Each $v_i \in DP$ is assigned a specific letter in a given alphabet that corresponds to the type of operation. For example, in Figure 3, the datapath {1'1, \$2, \$4, \$7, \$9, out} has the sequence $IAMMDO$.

The datapath itself does not capture the entirety of the dataflow. In order to capture as much of the dataflow as possible, three different datapaths are extracted, the maximum length path (DP_{max}), minimum length path (DP_{min}), and the longest path with the greatest number of unique components (DP_{alpha}). The functional component of the birthmark is formally defined as follows:

Definition 2.1. The functional component F of the birthmark B is $B_F(C) = \{DP_{max}, DP_{min}, DP_{alpha}\}$.

2) Structural Component: The second component corresponds to the structural aspect of the circuit using a molecular similarity matching techniques. Molecular similarity matching [15] is a technique used to compare the similarity of two molecules using fingerprints. A typical fingerprint is a bit-vector where each bit represents the presence or absence of a specific structural pattern in a molecule. The intuition is that the more patterns two molecules share in common, the more similar the molecules are to each other. The idea of fingerprinting is extended and applied to circuits.

The first step is to define the fingerprint for a circuit C. A fingerprint FP is a vector where each index i of FP represents a predefined subcomponents, $\{sc_1...sc_m\}$ where $sc_i \in FP$, $i = 1, 2, ..., m$. In order to capture the number of occurrences, a generic vector is used where $FP(C)_i$ is the

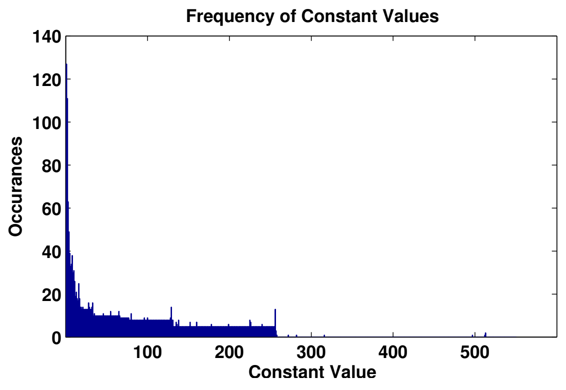

Fig. 5. Breakdown of occurrences of constants in 151 Verilog files.

number of occurrences of the subcomponent sc_i in C. Each sc_i is predetermined and represents a specific operation within the dataflow, such as adders, memory elements, shifters, etc. Figure 4 shows the fingerprinting process of a 8-bit counter.

Definition 2.2. The structural component S of B is $B_S(C) = FP(C) = \{\#sc_1....\#sc_m\}$.

3) Constant component: The third component represents constants instantiated in a design. Depending on the functionality of the circuit, designs with similar constants could suggest that there exists a relationship between the two. For example, typical baud rate constants can be a reasonable indicator that the design is or contains communication interfaces. Thus designs with similar constant values suggest that they could be related.

Since constants can take on a wide range of values, it is infeasible to create a feature vector that spans the entire constant space. Therefore, a binning approach is used. Figure 5 analyzed over 150 circuits from a wide range of sources including OpenCores, the current leading community for open source hardware IP cores. From the figure, most of the constant values range from between 0 and 256. In addition, small spikes in Figure 5 suggests that constants that are powers of two are fairly common. Taking the data into account, a bin is made for every constant from 0 to 256 as well as additional bins for each power of two greater than 256 and constants between them up to 2^{32}. Additional bins are added for don't-cares as

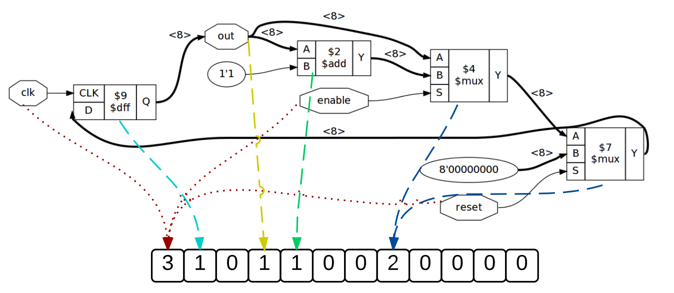

Fig. 4. The figure shows how the fingerprinting process works. Each position in the vector indicates the number of occurrence of a specific subcomponent

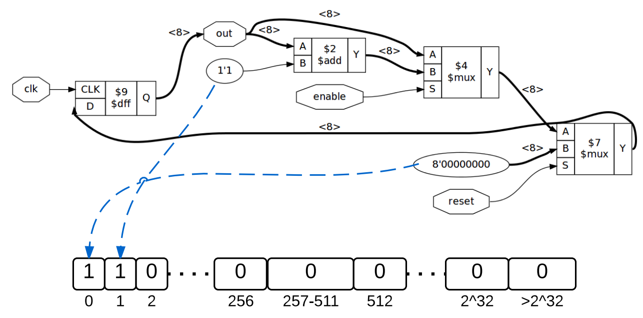

Fig. 6. Extraction of the constant component. The figure also shows the basic layout of the binning approach.

well as high-impedance inputs. Figure 6 shows the extraction of the constants as well as the layout of the bins used.

Definition 2.3. The constant component $Cnst$ of B, $B_{Cnst}(C) = \{b_1, ...b_n\}, k_i \in b_i$ where b_i is the ith bin and k_i is a constant found in C.

C. Computing Similarity Between Birthmarks

Computing the similarity between two birthmarks depends on the scheme of the birthmark. Since the birthmark consists of three different components, a separate method is needed for each distinct component.

1) Functional Similarity: The birthmark extracted from the functional birthmark consists of a variable-length sequence. One way to compute the similarity between two sequences is to use a sequence dynamic programming alignment technique such as Smith-Waterman. Sequence alignment methods are commonly used in bioinformatic as a way to find common regions or mutations in biological structures. In regards to the functional component, similar regions in the datapath can be located. In addition, local alignment techniques provide a way to see if the datapath is part of a larger one. A scoring matrix is also applied such the scores can be fine tuned for each pair of operation in the datapath. Operations that are more uncommon have a higher score when matched such as look-up table blocks. Furthermore, the matching score can be set higher for blocks that are similar in functionality. For example, multiplication by two can be rewritten by shifting to the left by one; however, the blocks are functionally different. As a result, the matching score for a multiplication to shift can have a lower matching or penalty score.

It is possible to compare the dataflow of two circuits directly using a subgraph isomorphism or a maximum common subgraph approach but the complexity of many graph-based algorithms are NP-complete. Using a sequence alignment method such as Smith-Waterman is only $O(nm)$ where n and m are the length of the two sequences instead of NP-complete. Moreover, a sequence alignment technique allows for mismatches and gaps to exist within a sequence that can represent errors the user made during the design phase or subtle difference based on individual design choices.

2) Structural and Constant Similarity: The structural and constant component of the birthmark are of the same representation where they enumerate the different structural features and characteristics of a circuit. Typical comparison methods for fingerprints use a similarity metric such as the Tanimoto coefficient due to chemical fingerprints consisting mainly of zeros [16].

Since the fingerprint in the structural component uses a fixed-length generic vector instead of a bit vector, a different similarity metric needs to be selected. Instead of Tanimoto's coefficient, a Euclidean distance metric is used to calculated the similarity between the two fingerprints.

3) Similarity Score: The scores of the different components need to be combined into a single score; however, the scale of the scores are different. The Euclidean metric is based on the distance between two fingerprints where the higher the distance, the more dissimilar the two components are. On the other hand, the score that the sequence alignment returns is based on the point-based scoring system used by the scoring matrix, rewarding points for correct matches, and penalizing for mismatches. The higher the score, the more similar the two components are.

The metrics in each component needs to be normalize such that they are between zero and one. The normalized Euclidean distance is subtracted from one so that a value towards one indicates similarity and a value towards zero indicates dissimilarity. Once the scores are normalized, they can be combined to form a single similarity score, sim with $0 \leq sim \leq 1$, that indicates how similar the two circuits are by averaging the scores. Many of the features and components can be weighted such that the more important component has a larger influence, but will be left for future work.

IV. Results and Analysis

A circuit similarity comparator tool was developed using the methods described above in C++ and Python. A Verilog design is taken as input, comparisons are made with the reference against the a database, and a list of similar circuits in ranked order is produced.

The tool uses Yosys [17] to extract the AST from the design. Currently, Yosys does not support VHDL but tools such as vhdl2verilog can be used to convert a VHDL file to Verilog. The Seqan library is used for pairwise sequence alignment of the functional birthmark. To evaluated the performance of the birthmarking approach, a database of circuits is constructed. The circuits in the database are extracted from a variety of sources from Opencores, [18], [19], [20], [21], and ranges from communications interfaces, such as UART and SPI, to filters and FFTs, totalling more than 150 circuits for initial testing purposes.

A. Performance

In order to suggest circuits that the designer can reuse, the comparison of two birthmarks has to be fast and efficient. If the comparison is too slow, the design may have changed significantly such that the results of the comparison is no longer meaningful to the designer. Real-time feedback whenever the design changes is desired.

While the performance of such a system is necessary to take into account, the focus of this paper is the identification of similar circuits. Many of these optimizations in regards to performance is left for future work. These tests were executed on a machine with an Intel Core2 Duo and 4GB of RAM.

1) Birthmark Extraction: Figure 7 show the total execution time to extract a birthmark from a Verilog design. The AST extraction from the Verilog design is all handled by the Yosys tools. Afterwards, the birthmark is extracted from the AST that Yosys generates. The execution time for several of the larger circuits such as the FFTs take close to ten seconds before the birthmark is extracted. However, based on the time it takes for a user to input and make changes on the design, this is

TABLE II
RANKING RESULTS FOR VARIOUS MODULES OF DIFFERENT TYPES

Score	mmuart* (COMM)	Score	cf_fft_256_18* (DSP)	Score	altera_sig_mult (ARTH)	Score	generic_dpram* (MEM)
84.09	mmuart_transceiver*	88.57	cf_fft_256_16*	86.21	altera_sig_altmult_add	92.84	ram_sp_sr_sw [19]
82.22	rtfSimpleUartRx*	86.69	cf_fft_256_8*	80.34	firfilter	90.02	ncsu_regfile [22]
81.44	tiny_spi*	48.64	cf_fft_512_18*	78.86	qmult*	88.08	ram_sp_ar_sw [19]
80.82	simple_spi_top*	41.78	cf_fft_512_16*	77.82	altera_unsig_altmult_accum	83.43	altera_true_dpram_sclk
80.60	rtfSimpleUart*	41.28	cf_fft_512_8*	76.11	firfilter2	80.63	behave1p_mem*
77.81	rtfSimpleUartTx*	35.16	qdiv2*	75.89	altera_unsigned_mult	79.78	behave2p_mem*
77.69	uart_rx_only	28.63	pipelined_fft_64*	72.34	IIR2_18bit_parallel	79.58	altera_ram_infer
77.41	uart_rx_only3 [23]	28.51	IIR6sos_18bit_fp	71.29	qmult*	79.58	altera_ram_dual
74.97	uart_rx_only2 [24]	28.21	dft_4_4_strm_dt [25]	71.28	mult_piped_8x8_2sC [20]	79.27	ncsu_fifo [22]
74.79	uart_simple [19]	27.33	qdiv*	71.28	mult_para_rc_8x8_2sC [20]	77.07	altera_single_port_ram

The IIR and CIC filters are examples from [21]. Circuits with the * beside them are from the Opencores database. The circuits with the altera heading are from [18]. The rest are circuits that had been manually designed.

a reasonable amount of time to take for extraction. Larger and more complex circuits can be addressed by looking for matches of the submodules in the lower levels of the hierarchy.

2) Circuit to database search: Results for comparing the birthmark of the reference circuit against a preprocessed database of existing designs can be seen in Figure 8. Since the structural and constant component of the birthmark are of a set size, the comparison between these components should not affect the overall performance of the comparison as the circuit becomes more complex; however, comparison of the functional component is based on the size of the two datapath sequences. Therefore, the execution time is heavily dependent on the size of the sequences. In order to scale to larger designs, hierarchical information of a design can be used of optimize the search space. The modules in the higher level of the hierarchy can be ignored if the similarity score shows little correlation when comparing the reference to the birthmark of the lower level modules. Decision trees or a relational data structure that groups similar designs together can help reduce the search space ignoring drastically different designs during the search phase.

B. Identifying similar circuits

Once the birthmarks are extracted, the similarity between circuits can be determined by directly comparing their birthmarks. By preprocessing the existing circuits into birthmarks ahead of time, the run-time comparison is significantly faster. Table II presents the results that the birthmark comparator returned. Four different reference circuits are observed, each in one of four domains: communications, digital signal processing, arithmetic, and memory hardware modules. Table II shows the top ten results returned by the ranking system. The score column represents the similarity score between the reference and the circuit in the database.

Overall, the ranking of similar existing designs to the reference turned out quite robust. Anomalies in the results can be seen and are to be expected since this is a similarity measurement. Looking at the communications column, most of the circuits found resembles different implementations of the UART controller from different designers. Certain SPI designs were ranked higher than UART, but since UART and SPI are both communication interfaces, their datapaths are fairly similar. With the DSP modules, the reference design is a radix 2 FFT with a transform size of 256 and the precision set to 18 bits. By observing the top five circuits, the birthmarking

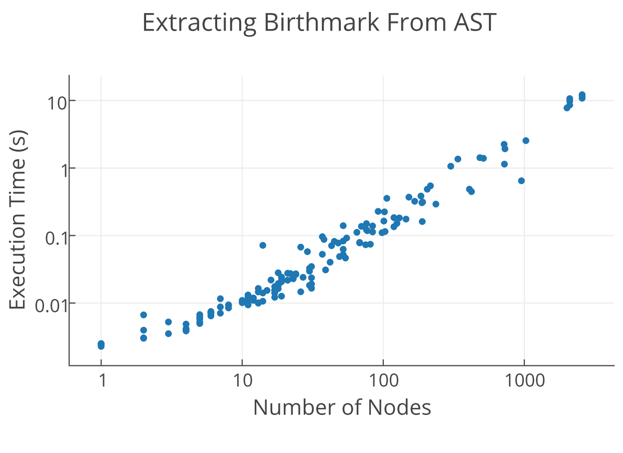

Fig. 7. Performance of extracting a birthmark from a verilog file. Execution time includes synthesis from Yosys and the proposed tools for extracting the birthmark from the AST

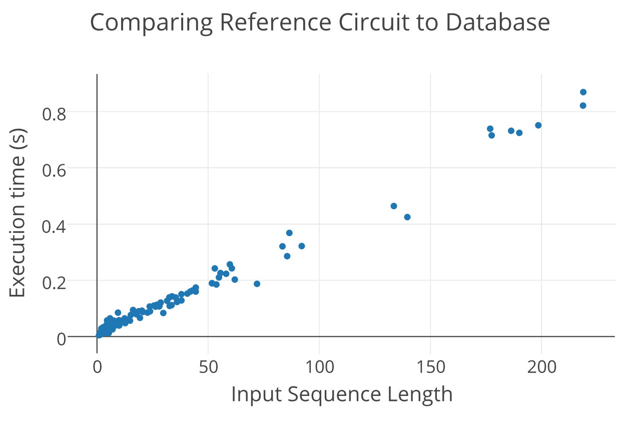

Fig. 8. Performance of comparing each circuit in the database to the database. Performance is dependent on the varying sequence length of the functional component

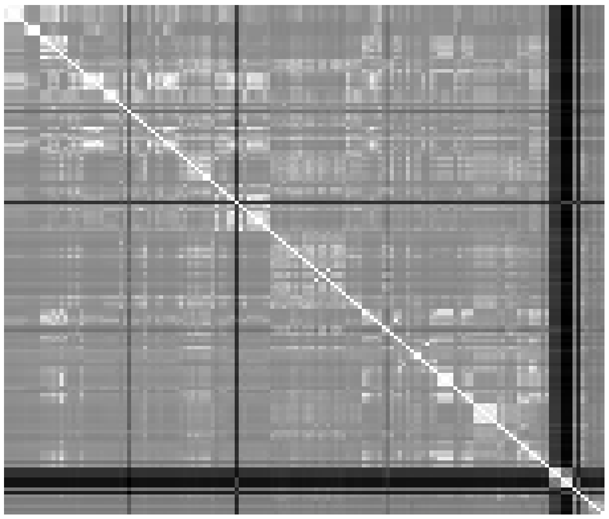

Fig. 9. Heatmap extracted by autocorrelating the circuits in the database. The dark regions indicates dissimilarity and the bright regions indicates similarity

approach is robust against similarly designed modules using different parameters. The tool was also able to find other implementations of FFTs from two different sources. With the multiplier from Altera, there were many instances of different multipliers found. Furthermore, it was able to extract some designs that commonly use multiplier blocks such as FIR and IIR filters. It was also able to detect its unsigned counterpart as well as a multipliers from several different sources. Lastly, using the generic dual-ported RAM module as reference, the tools were able to find many other memory modules from various sources as well as other memory-like modules such as a register file.

In Figure 9, a heatmap was extracted by autocorrelating the database of existing designs. The darker the regions is the less similar two circuits are and the brighter the region, the more similar the circuits are. Since the circuits are group together by what is believed to be similar circuits, most of the brighter regions are close to or near the diagonal. Some anomalies can be seen throughout the map and are explained in the next section.

C. Limitations

As noted before, there are certain limitations in which the birthmark does not capture all the information contained in a specific design. The birthmarking approach does a decent job at classifying circuits that are within or close to the same domain. This means that it can identify circuits that are closely related. For example, whenever a UART design is used as a reference, the top results are usually UART interfaces or SPI interfaces and can be seen for various UART or SPI circuits; however, this means that this approach does not handle intraclass comparisons very well. When trying to find similar UART designs, SPI designs may end up being one of the top results or vice versa.

Finding the superset of a module is also not very effective. For example, a FIFO design that uses the generic dual-ported RAM reference circuit is ranked at 93 out of 151 circuits even though it contains that specific design. The structural components are similar, but the number of each component is different, which make it appear different. Using a cosine similarity metric instead of a Euclidean could help alleviate this, Even so, this would be beneficial if the fingerprint size is much larger. Many circuits share a good amount of the predefined structural components and therefore the angles won't change significantly from circuit to circuit. Therefore, just the cosine similarity metric alone would not be able to efficiently distinguish the results.

It is important to note that the AST extracted from a HDL design depends on the design choices the designer makes. If the Verilog module is designed using a behavioral syntax, the the AST extracted will use behavioral datapath components and similarly the same for structural syntax. This means that two functionally equivalent circuits designed using both structural and behavioral Verilog will have completely different datapath because of the granularity of how the circuit was described. On the other hand, it can be argued that if it is common to have a circuit described in a specific syntax, then it is more likely that there will be an existing design described the same way. Furthermore, as the database of circuit grows, designs of both nature are more likely to exist so that when the designer compares his reference to the circuits in the database, there will likely be a close match.

V. Conclusion

A birthmarking technique for efficiently comparing two circuits was presented. This is necessary in order to suggest similar circuits for reuse such that the productivity of the designer can be improved. If circuits are suggested automatically for the user within the design scope, then the designer is more likely to reuse a design if it fits within their specifications. The system could also be used to build reliable hardware libraries by finding circuits similar in nature. Furthermore, with a centralized or even a local database of existing designs, it can create a richer experience for designers looking to use FPGAs to accelerate their application and to help broaden the hardware community as designers contribute and learn as a whole.

Many of the techniques described here are similar to birthmarking methods found in software birthmarks in which key characteristics inherent to a specific design are extracted. Key contributions in this paper are as followed:

1) A new design methodology is described by integrating reuse into the design environment in order to improve productivity.
2) A birthmarking technique was presented for hardware designs that captures both the functional and structural properties of a circuit into a compact representation.
3) Methods for comparing the similarity of hardware designs using the birthmarking technique was described

4) Preliminary experiments and results show that this approach yields promising results in comparing and ranking circuits using the birthmarking technique.

Future work could explore different birthmarking schemes for digital circuits which better captures the inherent characteristics of the birthmark. For example, exploring ways to better capture the underlying functionality regardless if the circuit is described behaviorally or structurally by looking into reverse engineering techniques. Furthermore, various optimizations and selection of parameters and weights can be explored in order to further improve performance and scalability of the methods. With a stable back-end, future motivation will be aimed towards analyzing productivity benefits from such a design flow as well as the design of a reusable circuit repository open to the hardware community.

References

[1] Altera, "White Paper Military Productivity Factors in Large FPGA Designs FPGA-based," no. July, pp. 1–7, 2008.

[2] J. Rodriguez-Andina, "Features, design tools, and application domains of FPGAs," *. . . , IEEE Transactions on*, vol. 54, no. 4, pp. 1810–1823, 2007.

[3] J. Whitham, "A Graph Matching Search Algorithm for an Electronic Circuit Repository," *Univ. of York*, 2004.

[4] M. Ohlrich, C. Ebeling, E. Ginting, and L. Sather, "SubGemini: Identifying SubCircuits using a Fast Subgraph Isomorphism Algorithm," *30th ACM/IEEE Design Automation Conference*, 1993.

[5] X. Shi, D. Zeng, Y. Hu, G. Lin, and O. R. Zaiane, "Enhancement of incremental design for FPGAs using circuit similarity," *Proceedings of the 12th International Symposium on Quality Electronic Design, ISQED 2011*, pp. 243–250, 2011.

[6] K.-h. Chang, D. A. Papa, I. L. Markov, and V. Bertacco, "Invers: an incremental verification system with circuit similarity metrics and error visualization," pp. 487–494, 2007.

[7] K. Zeng and P. Athanas, "Enhancing productivity with back-end similarity matching of digital circuits for IP reuse," *2013 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2013*, 2013.

[8] P. Subramanyan, N. Tsiskaridze, W. Li, A. Gascón, W. E. I. Y. Tan, A. Tiwari, N. Shankar, S. A. Seshia, and S. Malik, "Reverse Engineering Digital Circutis Using Structural and Functional Analyses," vol. 2, no. 1, pp. 63–80, 2014.

[9] W. Li, "Formal Methods for Reverse Engineering Gate-Level Netlists," 2013.

[10] T. Kakimoto and A. Monden, "Using Software Birthmarks to Identify Similar Classes and Major Functionalities WizStep SettingsTab," pp. 171–172, 2006.

[11] D. Kim, S.-j. Cho, S. Han, M. Park, and I. You, "Open Source Software Detection using Function-level Static Software Birthmark," *Journal of Internet Services and . . .* , vol. 4, no. November, pp. 25–37, 2014.

[12] Y. Chen, A. Narayanan, S. Pang, and B. Tao, "Malicioius Software Detection Using Multiple Sequence Alignment and Data Mining," *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, pp. 8–14, Mar. 2012.

[13] H. Kim and P. Li, "Polymorphic Attacks against Sequence-based Software Birthmarks."

[14] G. Myles and C. Collberg, "K-gram based software birthmarks," *Proceedings of the 2005 ACM symposium on Applied computing - SAC '05*, p. 314, 2005.

[15] V. Monev, "Introduction to similarity searching in chemistry," *MATCH Commun. Math. Comput. Chem*, vol. 51, pp. 7–38, 2004.

[16] (2011) Fingerprints- screening and similarity. [Online]. Available: http://www.daylight.com/dayhtml/doc/theory/theory.finger.html

[17] C. Wolf, "Yosys open synthesis suite," http://www.clifford.at/yosys/.

[18] Altera. [Online]. Available: https://www.altera.com/support/support-resources/design-examples/design-software/verilog.html

[19] Asic world. [Online]. Available: http://www.asic-world.com/examples/verilog/index.html

[20] Doulous. [Online]. Available: http://www.doulos.com/knowhow/verilog_designers_guide/models/

[21] Dsp examples. [Online]. Available: http://people.ece.cornell.edu/land/courses/ece5760/DE2/fpgaDSP.html

[22] Ncsu. [Online]. Available: http://www.ece.ncsu.edu/asic/lect_NTU/AppendixA.pdf

[23] Patchell ip archive. [Online]. Available: http://www.oldcrows.net/~patchell/IpArchive/

[24] Referencevoltage. [Online]. Available: http://referencevoltage.com/?p=54

[25] Spiral. [Online]. Available: http://www.spiral.net/hardware/dftgen.html